Lecture 8

COT4210 DISCRETE STRUCTURES

DR. MATTHEW B. GERBER

6/16/2016

PORTIONS FROM SIPSER, *INTRODUCTION TO THE THEORY OF COMPUTATION*, 3RD ED., 2013

Context-Free Grammars: Chomsky Normal Form

A simplified form for context-free grammars

- Useful for working with CFGs using algorithms

A CFG is in **Chomsky Normal Form** if:

- Every rule is of one of the following forms:
  - $A \rightarrow BC$
  - $A \rightarrow$ **a**
  - $S \rightarrow \varepsilon$
- Where *A*, *B* and *C* are variables, **a** is a terminal, and:
  - *S* is the starting variable
  - Neither *B* nor *C* are *S* (*A* can be)

Converting to CNF: 4-Step Process

1. **Add a new start variable.**
   It rewrites only to the old start variable:
   - $S_0 \rightarrow S$
2. **Get rid of rewrites to the empty string.**
   For every rewrite of variable $X \rightarrow \varepsilon$:
   - Remove the rule $X \rightarrow \varepsilon$
   - Find every instance of a variable Y being rewritten to anything involving X
   - Add a new rule rewriting Y to the same thing, but with X removed
3. **Get rid of unit rules.**
   For every rewrite of variables $X \rightarrow Y$:
   - Remove the rule $X \rightarrow Y$
   - Find every instance of Y being rewritten to anything
   - Add a new rule rewriting X to the same thing
4. **Convert all the remaining rules.**
   For every rule $X \rightarrow y_1y_2y_3\ldots y_n$:
   - Remove the rule $X \rightarrow y_1y_2y_3\ldots y_n$
   - Make new rules $X \rightarrow y_1X_1$, $X_1 \rightarrow y_2X_2$, $X_2 \rightarrow y_3X_3$, …, $X_{n-2} \rightarrow y_{n-1}y_n$
     - When making these rules, for every y_i that's a terminal:
       - Replace it with a new variable Y_i
       - Create new rule $Y_i \rightarrow y_i$

CNF Conversion Example

(Board work: 2.10)

Review: Stacks

A *stack* is a storage mechanism

- First-in, last-out
- *Push* data *onto* the top of the stack
- *Pop* data *from* the top of the stack
- Items below the top of the stack aren't accessible except by popping the top first

Pushdown Automata

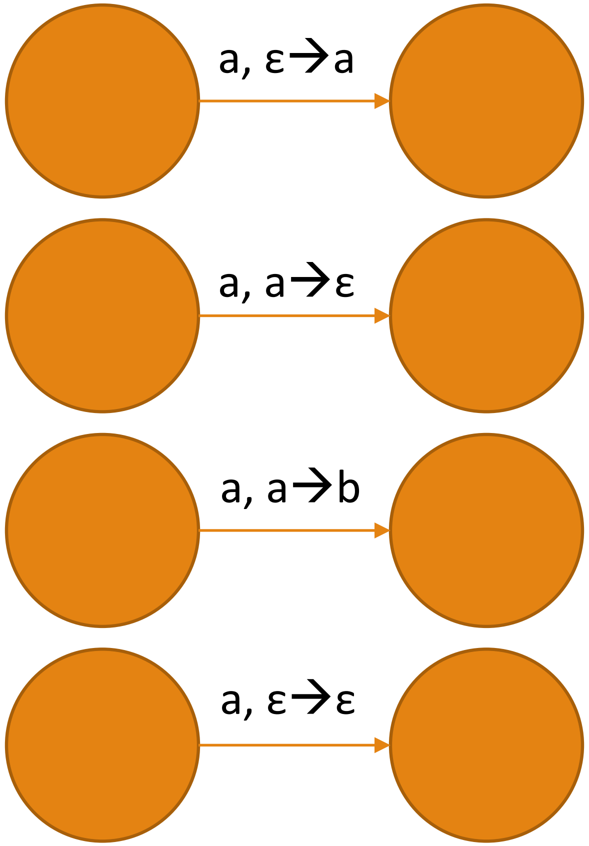
A *stack* is a storage mechanism

- First-in, last-out
- *Push* data *onto* the top of the stack
- *Pop* data *from* the top of the stack
- Items below the top of the stack aren't accessible except by popping the top first

A **pushdown automaton** is an NFA with a stack

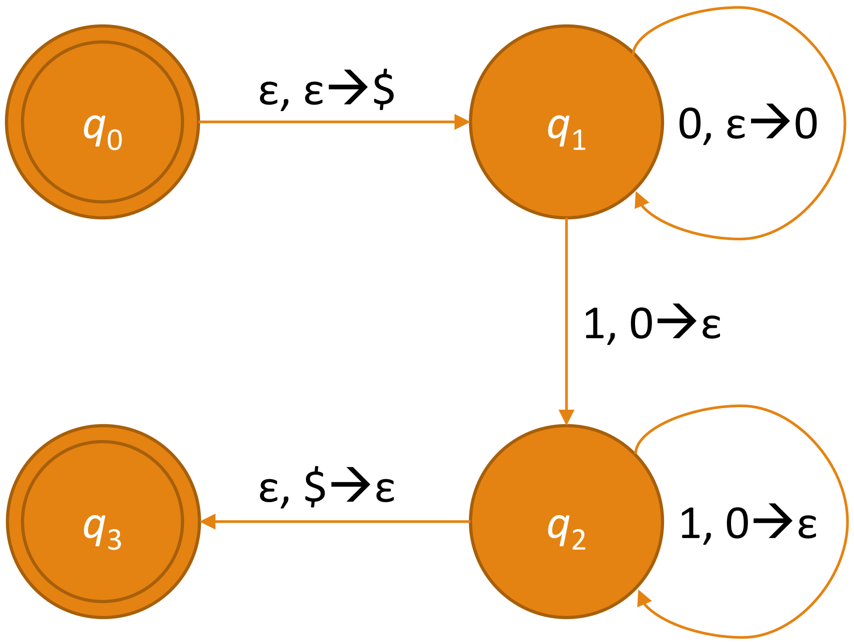
On any transition:

- Push…
- Pop…
- Both…
- …or neither



Recognizing a Familiar Language

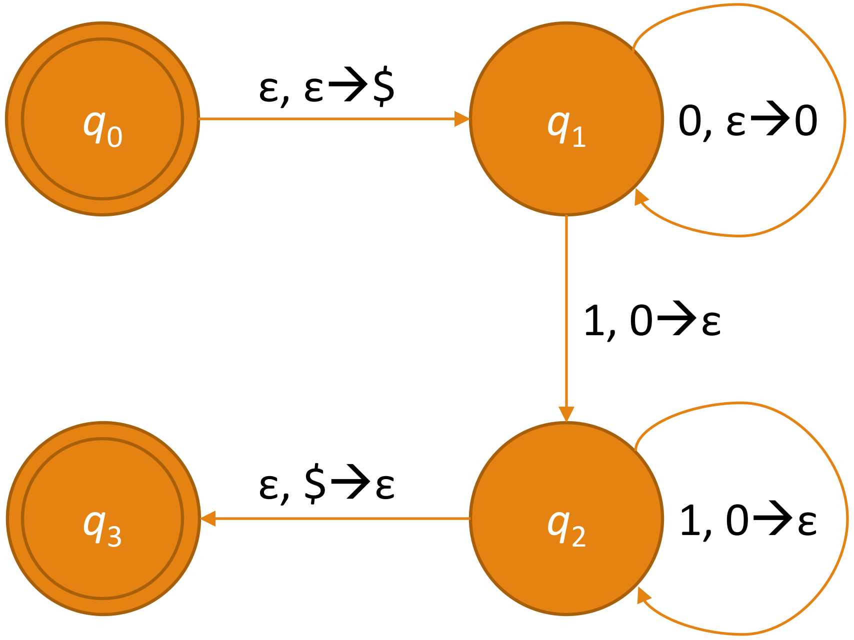
Can you tell what language this recognizes?



Recognizing a Familiar Language

Can you tell what language this recognizes?

$\{ 0^n 1^n, n \geq 0 \}$



Definition: Pushdown Automaton

A **pushdown automaton** is a 6-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ with:

- Q as the set of states,
- Σ as the input alphabet,
- Γ as the stack alphabet,
- $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ as the transition function,
- $q_0 \in Q$ as the start state, and
- $F \subseteq Q$ as the accept states.

It accepts a string w if:

- $w = w_1w_2\ldots w_n$, all $w_i \in \Sigma_\varepsilon$
  - (The usual string splitting)
- There's a state sequence $r_0, r_1, \ldots r_m \in Q$
  - (The usual state sequence)
- and a string sequence $s_0, s_1, \ldots s_m \in \Gamma^*$
  - (A sequence of stack contents)

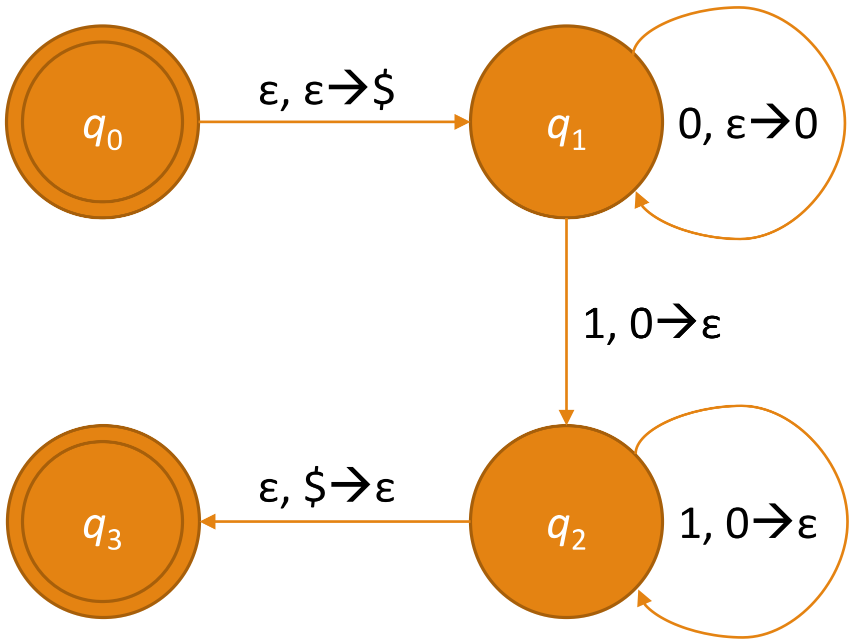
…so that $r_0 = q_0$, $s_0 = \varepsilon$, $r_m \in F$, and…

- For i from 0 to $m - 1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$,
  - (The usual transition mechanics…)
- …with $s_i = at$ and $s_{i+1} = bt$,
- …for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
  - (…with the stack transition mechanics added on)

Formalizing our First PDA

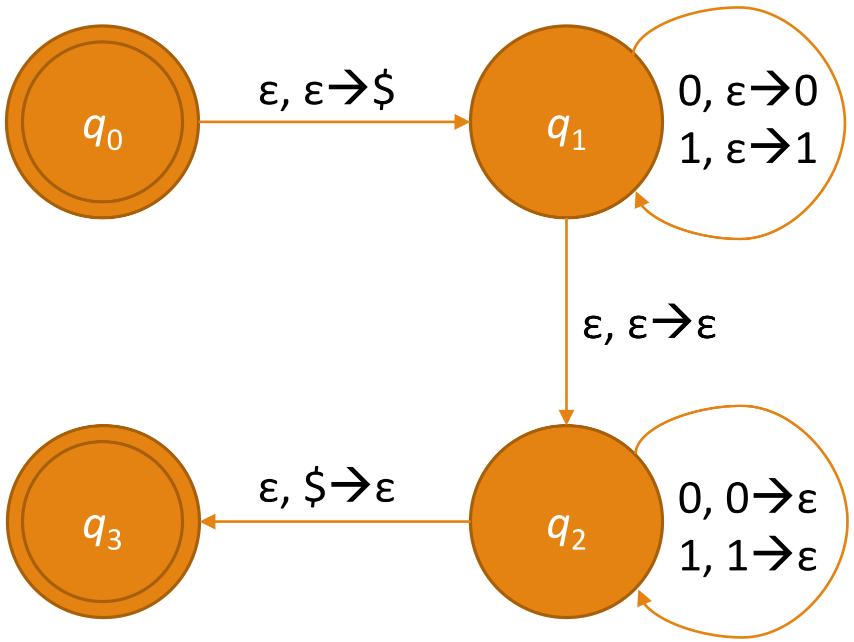
- $Q = \{ q_0, q_1, q_2, q_3 \}$
- $\Sigma = \{ 0, 1 \}$
- $\Gamma = \{ 0, \$ \}$
- $F = \{ q_0, q_3 \}$
- δ:

Input:	**0**			**1**			**ε**		
Stack:	**0**	**$**	**ε**	**0**	**$**	**ε**	**0**	**$**	**ε**
q_0									$\{(q_1, \$)\}$
q_1			$\{(q_1, 0)\}$	$\{(q_2, \varepsilon)\}$					
q_2				$\{(q_2, \varepsilon)\}$				$\{(q_3, \varepsilon)\}$	
q_3									



Another Familiar Language

Can you tell what language this recognizes?

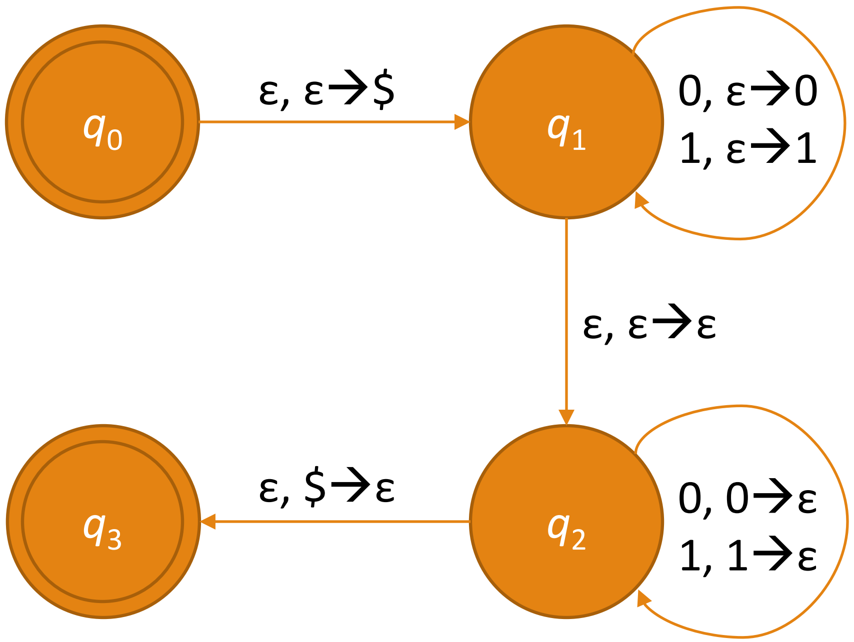


Another Familiar Language

Can you tell what language this recognizes?

$$\{ ww^R, w \geq \{0, 1\}^* \}$$

Notice that we use the power of nondeterminism to "guess" when we need to switch between the string and its reverse. This is common for PDAs.



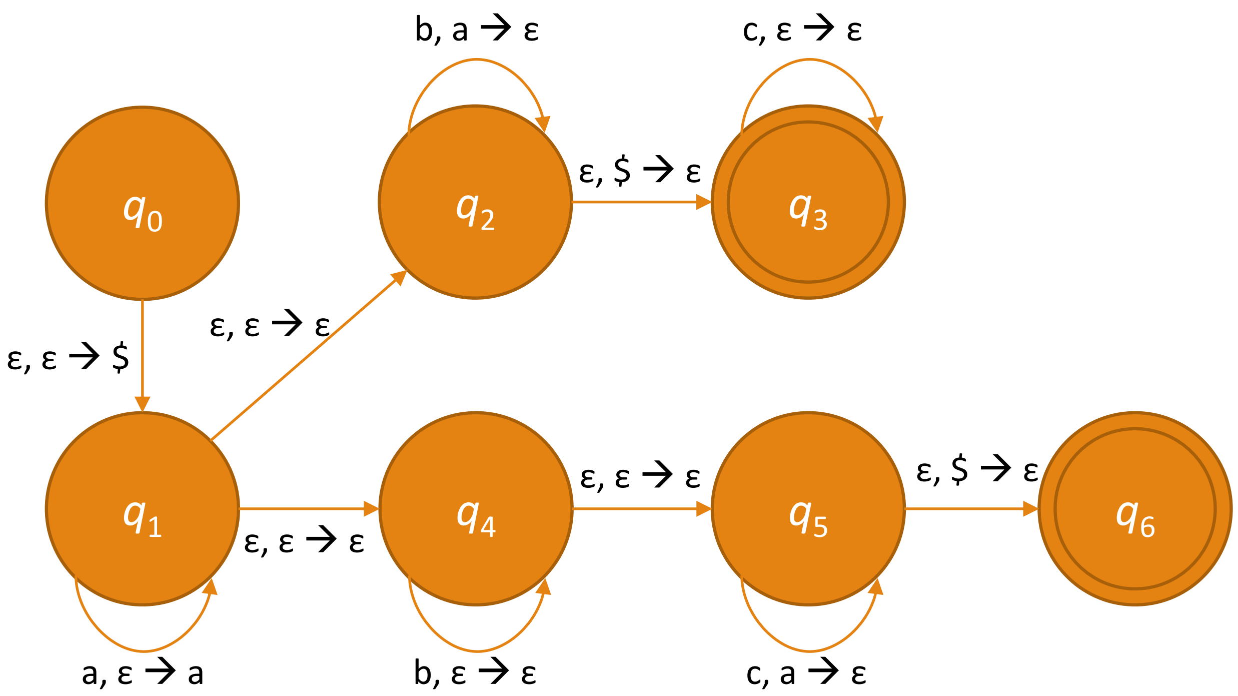
One More Language

This one recognizes:

$$\{\mathbf{a}^i\mathbf{b}^j\mathbf{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$$

We use nondeterminism *twice* here

- Once to decide whether we're matching the **b**'s or the **c**'s with the **a**'s
- In the case of matching **c**'s, to decide when to stop throwing away **b**'s and start processing **c**'s



Standard PDA Tricks, Part 1

The Stack Bottom Symbol

- A PDA doesn’t normally have any way to tell when the stack is empty
- Many PDAs push a unique “stack bottom” symbol – usually **$** - onto the stack as they begin execution
- This allows testing for an empty stack by looking for that same symbol

Pushing Strings

- We can push a string onto the stack just as easily as we can a symbol
- Imagine a sequence of empty-string transitions that each push a single symbol of the string
- From now on, we’ll allow ourselves to write transitions as though they were pushing strings
- When we do this, *we push the **last** symbol of the string **first*** – if we are pushing **xyz**, we push **z** then **y** then **x**

Recognizing CFLs: The Plan

PDAs are equivalent in power to CFGs

- (admit it, you're not very surprised)
- As always, two things to prove
- First, let's prove that a PDA can recognize any CFL

We *heavily* exploit the nondeterminism of PDAs

- Remember that a derivation in a CFG is a sequence of substitutions
- We want to accept a string if any derivation of it in the grammar exists
- We don't have to figure out which path to take – since a PDA is non-deterministic, it takes them all at once

We use the stack to "walk" the string

- For every variable, try every possible substitution
- For every terminal, try to find a match

Recognizing CFLs: General Method

- Push stack-bottom symbol **$**
- Repeat forever:
  - Pop the stack, and switch on the result:
    - For variable *A*:
      - New non-deterministic branch for each rule with *A* as the LHS
      - On each branch:
        - Push the RHS
    - For terminal **b**:
      - Read the next input symbol
      - If the next input symbol is **b**, continue
      - If not, reject on this branch
    - For stack-bottom symbol **$**:
      - Enter the accept state
        - Accept the input **if it's all been read**

Recognizing CFLs: Construction

Given a CFL $\boldsymbol{G} = (\boldsymbol{V}, \Sigma_C, \boldsymbol{R}, \boldsymbol{S})$:

- ***V*** is the ***variables***
- **Σ** is the ***terminals***
- ***R*** is the ***rules***
- $\boldsymbol{S} \in \boldsymbol{V}$ is the ***start variable***

Create an NFA ***N*** with:

- $Q = \{ q_0, q_{loop}, q_{accept} \}$
- $\Sigma = V \cup \Sigma_C$
- $F = \{ q_{accept} \}$
- δ:
  $\delta(q_0, \varepsilon, \varepsilon) = \{ (q_{loop}, S\$) \}$
  $\delta(q_{\text{loop}}, \mathbf{a} \in \Sigma_C, \mathbf{a}) = \{ (q_{loop}, \varepsilon) \}$
  $\delta(q_{\text{loop}}, \varepsilon, A \in V) =$
  $\{(q_{loop}, RHS(r)) \mid r \in R,\ LHS(r) = A\}$
  $\delta(q_{\text{loop}}, \varepsilon, \$) = \{ (q_{accept}, \varepsilon) \}$
  ∅ otherwise

Construction Examples

(Board work)

Standard PDA Tricks, Part 2

Single Accept State

- Just as easy as with an ordinary NFA
- Empty string, stack no-operation transitions from what would otherwise be accept states to a unified accept state

Empty Stack Before Accepting

- Get to a single-accept state
- Make it a non-accept state
- Add an empty-string self-loop that pops anything except **$** off the stack
- Add an empty-string transition, that pops **$**, to a new accept state

Always Push Or Pop, Never Both

- Rewrite transitions that **push *and* pop** to **pop *then* push**, using a new middle state and an empty-string transition
- Rewrite transitions that neither **push *nor* pop** to **push *then* pop** a dummy symbol, again using a new middle state and an empty-string transition

Grammars for PDAs: Modifying the PDA

Let's show that a CFG can generate the language of any PDA

- Take a PDA *P*
- It suffices to construct a CFG *G* that generates the language it accepts

First, let's modify it as we discussed on the last slide. Let P_G be *P* modified so that:

- It has a single accept state
- It empties its stack before it accepts
- Every transition either pushes or pops; not both and not neither
- Since we know P_G accepts equivalently to *P*, it suffices to construct a CFG *G* that generates the language of P_G

It suffices in turn to construct a grammar G, and show that ***G* generates a string *s* if *s* causes P_G to go from its start state to its accept state**

Grammars for PDAs: Stack-Preserving Transitions

Our construction is, fundamentally, as follows:

- Consider **every** pair of states (q_w, q_z) in P_G
- Create a variable V_{wz} derivable to all the strings that:
  - Take the machine from q_w to q_z
  - Leave the stack empty if it starts empty
- Note that the second part is really just "leave the stack like we found it"
  - If we leave the stack empty given that it starts empty, then we will leave it containing string *s* given that it started containing string *s*

Grammars for PDAs: The Induction Plan

Remember the restrictions on P_G:

- Single accept state
- Empty stack before accepting
- Always push or pop, never both or neither

Consider any string s so that s takes P_G from q_w to q_z preserving stack emptiness

- The *first* move *must* be a push
- The *last* move *must* be a pop

If the symbol **popped at the end** is the **same symbol pushed at the beginning**, the stack **might** only be empty at the beginning and end

- V_{wz} = **a**V_{xy}**b** where:
  - **a** is the input read along with that first push
  - **b** is the input read along with that last pop
  - x is the state just after w
  - y is the state just before z

Otherwise, the stack is empty **at some point in between**

- $V_{wz} = V_{wx}V_{xz}$ where x is the state at that point

Grammars for PDAs: Construction

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted so that it:

- Has a single accept state
- Empties its stack before accepting
- Always pushes or pops on a transition, never both or neither

We have rules to:

1. Handle the degenerate case of a state transitioning to itself
2. Handle the transitive transition case
3. Handle the push-pop case

Let *G* be a CFG and construct its rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 1 Basis

We want to show that **if** V_{wz} generates string s, **then** s takes P_G from q_w to q_z preserving stack emptiness.

We show this by induction on the number of steps in the derivation of s.

Basis: The derivation has one step. Therefore, the RHS cannot have variables. The only rules without variables on the RHS in G are rules of the form $V_{ww} \rightarrow \varepsilon$. Clearly ε takes P_G from q_w to q_w preserving stack emptiness, as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 1 Setup

Induction Hypothesis: If V_{wz} generates string s by a derivation with k or fewer steps, $k \geq 1$, then s takes P_G from q_w to q_z preserving stack emptiness.

Induction: Show that if V_{wz} generates string s by a derivation with $k + 1$ steps, then s takes P_G from q_w to q_z preserving stack emptiness.

Consider $V_{wz} \rightarrow^* s$ in $k + 1$ steps. The first step must be either $V_{wz} \rightarrow V_{wx}V_{xz}$ or $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 1 Case 1

Induction: Show that if V_{wz} generates string s by a derivation with $k + 1$ steps, then s takes P_G from q_w to q_z preserving stack emptiness.

Consider $V_{wz} \rightarrow^* s$ in $k + 1$ steps. The first step must be either $V_{wz} \rightarrow V_{wx}V_{xz}$ or $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$.

If it's $V_{wz} \rightarrow V_{wx}V_{xz}$, then:

- $s = rt$ so that $V_{wx} \rightarrow^* r$ and $V_{xz} \rightarrow^* t$, both in k or fewer steps.
- Therefore by the induction hypothesis, r takes P_G from q_w to q_x and t takes P_G from q_x to q_z, both preserving stack emptiness.
- Therefore, rt takes P_G from q_w to q_z preserving stack emptiness. Since $rt = s$, so does s, as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 1 Case 2

If it's $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$, then:

- $s = \mathbf{a}t\mathbf{b}$ so that $V_{xy} \rightarrow^* t$ in k or fewer steps.
- Therefore by the induction hypothesis, t takes P_G from q_x to q_y preserving stack emptiness.
- By part 3 of our construction, since $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$ is a rule, then for some stack symbol **u**, $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$ and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε).
- Therefore, P_G can:
  - Read **a** and push **u** to go from q_w to q_x
  - Use t to go from q_x to q_y with only **u** left on the stack
  - Read **b** and pop **u** to go from q_y to q_z
- …which leaves the stack empty, and we have completed our transition as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

We're almost done.

REALLY.

Grammars for PDAs: Showing It Works – Direction 2 Basis

We want to show that **if** s takes P_G from q_w to q_z preserving stack emptiness, **then** V_{wz} generates string s. We show this by induction on the number of steps in P_G's computation from q_w to q_z.

Basis: The computation has no steps.

- Therefore, it starts and ends at the same state w.
- Therefore, we need V_{ww} to generate s.
- In 0 steps, P_G can't read anything, so $s = \varepsilon$.
- By part 1 of our construction, we have $V_{ww} \rightarrow \varepsilon$ as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 2 Setup

Induction Hypothesis: If *s* takes P_G from q_w to q_z preserving stack emptiness by a computation with *k* or fewer steps, $k \geq 0$, then V_{wz} generates string *s*.

Induction: Show that if *s* takes P_G from q_w to q_z preserving stack emptiness by a computation with $k + 1$ steps, then V_{wz} generates string *s*.

Suppose *s* takes P_G from q_w to q_z preserving emptiness by a computation with $k + 1$ steps.

Then either the stack becomes empty somewhere in between q_w and q_z, or the stack is empty only at the beginning and end of this computation.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let *G* be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 2 Case 1

Either the stack becomes empty somewhere in between q_w and q_z, or the stack is empty only at the beginning and end of this computation.

If the stack becomes empty between them:

- Let q_x be the state where it does so.
- Then the computations from q_w to q_x, and q_x to q_z, have k or fewer steps.
- Let $s = rt$ where r takes P_G from q_w to q_x and t takes P_G from q_x to q_z.
- By the induction hypothesis, $V_{wx} \rightarrow^* r$ and $V_{xz} \rightarrow^* t$.
- By part 2 of our construction, $V_{wz} \rightarrow V_{wx}V_{xz}$.
- Then $V_{wz} \rightarrow^* rt$, and since $rt = s$, $V_{wz} \rightarrow^* s$ as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Grammars for PDAs: Showing It Works – Direction 2 Case 2

If the stack *doesn't* become empty in between q_w and q_z:

- Observe that the symbol **u** that is pushed at the first move must also be popped at the last move.
- Let **a** be the input read in the first move, and **b** be the input read in the last move, and *t* be the part of s between them, so that s = **a**t**b**.
- Let q_x be the state just after q_w and q_y be the state just before q_z.
- t takes P_G from q_x to q_y in $(k - 1)$ steps. Therefore, by the induction hypothesis, V_{xy} generates t.
- By the third part of our construction, $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$. So $V_{wz} \rightarrow^* \mathbf{a}t\mathbf{b}$.
- Since $s = \mathbf{a}t\mathbf{b}$, $V_{wz} \rightarrow^* s$ as desired.

Let PDA $P_G = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ restricted to single accept state, empty stack before accept, and always-push-or-pop.

Let G be a CFG with rules as follows:

1. For all $w \in Q$
   - Add rule $V_{ww} \rightarrow \varepsilon$
2. For all $w, x, z \in Q$,
   - Add rule $V_{wz} \rightarrow V_{wx}V_{xz}$
3. For all $w, x, y, z \in Q$, $\mathbf{u} \in \Gamma$, and $\mathbf{a}, \mathbf{b} \in \Sigma_\varepsilon$:
   - **If** $\delta(w, \mathbf{a}, \varepsilon)$ contains $(x, \mathbf{u})$
     and $\delta(y, \mathbf{b}, \mathbf{u})$ contains (z, ε)
     then add rule $V_{wz} \rightarrow \mathbf{a}V_{xy}\mathbf{b}$

Construction Examples

Construction Examples

No.

PDA-CFG Equivalence

We've shown that:

- Any context-free grammar's language can be recognized by a pushdown automaton
- Any pushdown automaton's language can be generated by a context-free grammar

PDAs and CFGs are equal in power. So we can now say all of the following:

- **A language is context-free if and only if a context-free grammar generates it.**
- **A language is context-free if and only if a pushdown automaton recognizes it.**
- **A context-free grammar generates a language if and only if a pushdown automaton recognizes it.**

A Corollary

We’ve just proven that PDAs recognize context-free languages.

- But a PDA is just an NFA with a stack.
- It can ignore its stack just like an NFA can ignore nondeterminism.

A Corollary

We've just proven that PDAs recognize context-free languages.

- But a PDA is just an NFA with a stack.
- It can ignore its stack just like an NFA can ignore nondeterminism.

Every regular language is also a context-free language.

Next Time: Deterministic PDAs, Non-CFLs, and More Pigeons