Quiz 1 Review
CSCI 338
Quiz 1 Logistics

1. During class on Thursday 9/29.
2. You can bring your book and any notes you would like, but no electronic devices.
3. You may assume anything proven in class or on homeworks.
4. Four questions:
   1) Show a language is regular.
   2) Show a language is not regular x2.
   3) Conceptual question x1.
Regular Languages

Assuming the alphabet is \{0,1\}, show that the following language is regular:

\[ L = \{ \omega : \text{every 0 is immediately followed by one or more 1} \} \]
Regular Languages

Assuming the alphabet is \{0,1\}, show that the following language is regular:
\[ L = \{ \omega: \text{every 0 is immediately followed by one or more 1} \} \]
Non-Regular Languages

Show that the following language is not regular:

\[ L = \{a^n b^m c^{n+m} : n, m \geq 0\} \]
Non-Regular Languages

Show that the following language is not regular:

\[ L = \{ a^n b^m c^{n+m} : n, m \geq 0 \} \]

Proof: Suppose \( L \) is regular. Let \( p \) be the number from the pumping lemma.

Consider \( s = a^p b^p c^{2p} \).

Since \( s \in L \) and \( |s| \geq p \), the conditions of the pumping lemma must hold for \( s = xyz \).

\( y = a^k \) for some \( k > 0 \), since \( |xy| \leq p \)

\[ \Rightarrow s = a^{p-k} a^k b^p c^{2p} \]

Consider the string \( s' = xy^2z = a^{p-k} a^{2k} b^p c^{2p} \)

The number of \( a \)'s and \( b \)'s = \( p - k + 2k + p = 2p + k \neq 2p \) (the number of \( c \)'s), since \( k > 0 \). \( \Rightarrow s' \notin L \), which is a contradiction of the pumping lemma.

Therefore, the language is not regular.
Non-Regular Languages

Show that the following language is not regular:

\[ L = \{ \omega : \omega \text{ contains an equal number of 0’s and 1’s} \} \]
Non-Regular Languages

Show that the following language is not regular:
\[ L = \{ \omega : \omega \text{ contains an equal number of } 0\text{'s and } 1\text{'s} \} \]

**Proof 1:** Suppose \( L \) is regular. Let \( p \) be the number from the pumping lemma. Consider \( s = 0^p 1^p \).

Since \( s \in L \) and \( |s| \geq p \), the conditions of the pumping lemma must hold for \( s = xyz \).

\[ y = 0^k \] for some \( k > 0 \), since \( |xy| \leq p \)

\[ \Rightarrow s = 0^{p-k} 0^k 1^p \]

Consider the string \( s' = xy^2z = 0^{p-k} 0^{2k} 1^p \)

The number of 0's = \( p - k + 2k = p + k \neq p \) (the number of 1's), since \( k > 0 \). \( \Rightarrow s' \notin L \), which is a contradiction of the pumping lemma.

Therefore, the language is not regular.
Non-Regular Languages

Show that the following language is not regular:

\[ L = \{ \omega : \omega \text{ contains an equal number of 0’s and 1’s} \} \]

**Proof 2:**

\[ L \cap 0^*1^* = \{0^n1^n : n \geq 0 \} \]

We know that \( 0^*1^* \) is regular and that \( \{0^n1^n : n \geq 0 \} \) is not regular. If \( L \) were regular, then we would have:

\[ \text{regular} \cap \text{regular} = \text{non-regular}. \]

That cannot happen, since the intersection of regular languages is regular, so \( L \) is not regular.
Beyond Regular - CFL/PDA
CSCI 338
Context-Free Grammar (CFG)

\[ G_1: \quad A \to 0A1 \]
\[ A \to B \]
\[ B \to \varepsilon \]

- Analogous to regular expressions (instructions for building strings).
- Enables languages with nested/recursive features.
- More powerful than regular languages.
Context-Free Grammar (CFG)

\[ G_1: \]

\[ A \rightarrow 0A1 \]

\[ A \rightarrow B \]

\[ B \rightarrow \varepsilon \]

Variables
Context-Free Grammar (CFG)

\[ G_1: \]

\[
\begin{align*}
A & \rightarrow 0A1 \\
A & \rightarrow B \\
B & \rightarrow \varepsilon
\end{align*}
\]
Context-Free Grammar (CFG)

\[ G_1: \]
\[ A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

Terminals (alphabet)
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \] Substitution
\[ A \rightarrow B \] Rules
\[ B \rightarrow \varepsilon \]
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
1. Write down the start variable.
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
Context-Free Grammar (CFG)

\[ G_1: \begin{align*}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \varepsilon
\end{align*} \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
Context-Free Grammar (CFG)

\[ G_1: \begin{align*}
A &\to 0A1 \\
A &\to B \\
B &\to \varepsilon
\end{align*} \quad A \to 0A1 \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

$G_1$: $A \rightarrow 0A1$

A $\rightarrow$ 0A1

A $\rightarrow$ B

B $\rightarrow$ $\epsilon$

String generation process:

1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

\[ G_1: \begin{align*}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \varepsilon
\end{align*} \]

\[ A \rightarrow 0A1 \rightarrow 00A11 \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

\[ G_1: \]
\[
A \rightarrow 0A1 \\
A \rightarrow B \\
B \rightarrow \varepsilon
\]

\[ A \rightarrow 0A1 \\
\quad \rightarrow 00A11
\]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

\[ G_1: \]

- \[ A \rightarrow 0A1 \]
- \[ A \rightarrow B \]
- \[ B \rightarrow \varepsilon \]

- \[ A \rightarrow 0A1 \]
- \[ A \rightarrow 00A11 \]
- \[ A \rightarrow 000A111 \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

\[
G_1:\begin{align*}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \varepsilon
\end{align*}
\]

\[
A \rightarrow 0A1 \\
\rightarrow 00A11 \\
\rightarrow 000A111 \\
\rightarrow 000B111
\]
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.
Context-Free Grammar (CFG)

\[ G_1: \]
\[
A \rightarrow 0A1 \\
A \rightarrow B \\
B \rightarrow \varepsilon
\]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

\[
A \rightarrow 0A1 \\
\quad \rightarrow 00A11 \\
\quad \rightarrow 000A111 \\
\quad \rightarrow 000B111 \\
\quad \rightarrow 000\varepsilon111 = 000111
\]
Context-Free Grammar (CFG)

\[ G_1: \quad A \rightarrow 0A1 \]
\[ A \rightarrow B \]
\[ B \rightarrow \varepsilon \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

Language of a CFG: Set of all strings that can be generated by a CFG.
Context-Free Grammar (CFG)

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

Language of a CFG: Set of all strings that can be generated by a CFG.
Context-Free Grammar (CFG)

\[ \{0^n1^n : n \geq 0 \} \]

\[ \begin{align*}
G_1 : & \quad A \rightarrow 0A1 \\
& \quad A \rightarrow B \\
& \quad B \rightarrow \epsilon
\end{align*} \]

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

Language of a CFG: Set of all strings that can be generated by a CFG.
Context-Free Grammar (CFG)

String generation process:
1. Write down the start variable.
2. For some variable written down, find a rule that starts with the variable and replace it with the right hand side of rule.
3. Repeat step 2 until no variables remain.

Language of a CFG: Set of all strings that can be generated by a CFG.
Context-free language: Any language generated by some CFG.
Language Hierarchy

- Context-Free Languages
- Regular Languages
- All Languages
Language Hierarchy

\[ \{0^n1^n : n \geq 0\} \]
Pushdown Automata (PDA)

NFA (not DFA) with a stack.
Pushdown Automata (PDA)

NFA (not DFA) with a stack.

Stack Review:
- “Last in, first out”
- Operations:
  - push (put an item on the top of the stack),
  - pop (take the top item off the stack)
Pushdown Automata (PDA)

NFA (not DFA) with a stack.

Stack Review:
- “Last in, first out”
- Operations:
  - push (put an item on the top of the stack),
  - pop (take the top item off the stack)

```
stack.push(1)
```
Pushdown Automata (PDA)

NFA (not DFA) with a stack.

Stack Review:
• “Last in, first out”
• Operations:
  • push (put an item on the top of the stack),
  • pop (take the top item off the stack)

```
stack.push(1)
stack.push(0)
```
Pushdown Automata (PDA)

NFA (not DFA) with a stack.

Stack Review:
- “Last in, first out”
- Operations:
  - push (put an item on the top of the stack),
  - pop (take the top item off the stack)

```
stack.push(1)
stack.push(0)
stack.push(0)
```
Pushdown Automata (PDA)

NFA (not DFA) with a stack.

Stack Review:
- "Last in, first out"
- Operations:
  - push (put an item on the top of the stack),
  - pop (take the top item off the stack)

```python
stack.push(1)
stack.push(0)
stack.push(0)
stack.pop()  # returns 0
```
Pushdown Automata (PDA)
Pushdown Automata (PDA)

Transition Notation:

Input, Stack character, pop → Stack push
Pushdown Automata (PDA)

Transition Notation:

<table>
<thead>
<tr>
<th>Input character</th>
<th>Stack pop</th>
<th>Stack push</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon, \epsilon \rightarrow $</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,0 \rightarrow \epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\epsilon, $ \rightarrow \epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, \epsilon \rightarrow 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,0 \rightarrow \epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\epsilon, $ \rightarrow \epsilon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If ‘$a’ is read from input and ‘$b’ is on top of stack, pop ‘$b’ and push ‘$c’.
Pushdown Automata (PDA)

Transition Notation:

<table>
<thead>
<tr>
<th>Input character</th>
<th>Stack pop</th>
<th>Stack push</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon, \varepsilon \rightarrow $$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$1,0 \rightarrow \varepsilon$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$\varepsilon, $ \rightarrow \varepsilon$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$0, \varepsilon \rightarrow 0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$1,0 \rightarrow \varepsilon$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
</tbody>
</table>

$a, b \rightarrow c$ If ‘$a$’ is read from input and ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.

$\varepsilon, b \rightarrow c$ Without reading input, if ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.
Pushdown Automata (PDA)

Transition Notation:

<table>
<thead>
<tr>
<th>Input character</th>
<th>Stack pop</th>
<th>Stack push</th>
</tr>
</thead>
<tbody>
<tr>
<td>a,\ e \rightarrow c</td>
<td>If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.</td>
<td></td>
</tr>
<tr>
<td>\e, b \rightarrow c</td>
<td>Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.</td>
<td></td>
</tr>
<tr>
<td>a, \e \rightarrow c</td>
<td>If ‘a’ is read, push ‘c’.</td>
<td></td>
</tr>
</tbody>
</table>
Pushdown Automata (PDA)

Transition Notation:
- $A, B \rightarrow C$: If 'A' is read from input and 'B' is on top of stack, pop 'B' and push 'C'.
- $\varepsilon, B \rightarrow C$: Without reading input, if 'B' is on top of stack, pop 'B' and push 'C'.
- $A, \varepsilon \rightarrow C$: If 'A' is read, push 'C'.
- $A, B \rightarrow \varepsilon$: If 'A' is read and 'B' is on top of stack, pop 'B'.

Input | Stack | pop | Stack | push
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $s = 0011$
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $s = 0011$
Pushdown Automata (PDA)

- $, \varepsilon \rightarrow \$
- $0, \varepsilon \rightarrow 0$
- $1,0 \rightarrow \varepsilon$
- $\varepsilon, \$, $\rightarrow \varepsilon$

- $a, b \rightarrow c$ If ‘$a$’ is read from input and ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.
- $\varepsilon, b \rightarrow c$ Without reading input, if ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.
- $a, \varepsilon \rightarrow c$ If ‘$a$’ is read, push ‘$c$’.
- $a, b \rightarrow \varepsilon$ If ‘$a$’ is read and ‘$b$’ is on top of stack, pop ‘$b$’.

$s = 0011$
Pushdown Automata (PDA)

\[\epsilon, \epsilon \rightarrow \$\]
\[1,0 \rightarrow \epsilon\]
\[\epsilon, \$ \rightarrow \epsilon\]

\[0, \epsilon \rightarrow 0\]
\[1,0 \rightarrow \epsilon\]

\[a, b \rightarrow c\] If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
\[\epsilon, b \rightarrow c\] Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
\[a, \epsilon \rightarrow c\] If ‘a’ is read, push ‘c’.
\[a, b \rightarrow \epsilon\] If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack:

\[s = 0011\]
Pushdown Automata (PDA)

If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

If ‘a’ is read, push ‘c’.

If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack:

\[
\begin{array}{c}
st = 0011 \\
\end{array}
\]
Pushdown Automata (PDA)

\[
\begin{align*}
\varepsilon, \varepsilon & \rightarrow \$ \\
1,0 & \rightarrow \varepsilon \\
\varepsilon, \$ & \rightarrow \varepsilon
\end{align*}
\]

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

\[
\begin{align*}
\varepsilon, b & \rightarrow c \\
\end{align*}
\]
Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

\[
\begin{align*}
a, \varepsilon & \rightarrow c \\
a, b & \rightarrow \varepsilon
\end{align*}
\]
If 'a' is read, push 'c'. If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack:

\[
0 \\
\$
\]

\[
s = 0011
\]
If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

If ‘a’ is read, push ‘c’.

If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.
Pushdown Automata (PDA)

- $\varepsilon, \varepsilon \rightarrow \$\ $
- $1,0 \rightarrow \varepsilon$
- $\varepsilon, \$ \rightarrow \varepsilon$

- $0, \varepsilon \rightarrow 0$
- $1,0 \rightarrow \varepsilon$

- $a, b \rightarrow c$  If ‘$a$’ is read from input and ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.
- $\varepsilon, b \rightarrow c$  Without reading input, if ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.
- $a, \varepsilon \rightarrow c$  If ‘$a$’ is read, push ‘$c$’.
- $a, b \rightarrow \varepsilon$  If ‘$a$’ is read and ‘$b$’ is on top of stack, pop ‘$b$’.

Stack: $0 0 0 1 1$
Pushdown Automata (PDA)

- $\varepsilon, \varepsilon \rightarrow \$$
- $1, 0 \rightarrow \varepsilon$
- $\varepsilon, \$$ \rightarrow \varepsilon$

- $a, b \rightarrow c$ If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- $\varepsilon, b \rightarrow c$ Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- $a, \varepsilon \rightarrow c$ If ‘a’ is read, push ‘c’.
- $a, b \rightarrow \varepsilon$ If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack: $\text{s} = 0011$
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack:

$$s = 0011$$
Pushdown Automata (PDA)

- $\varepsilon, \varepsilon \rightarrow \$ $
- $1,0 \rightarrow \varepsilon$
- $\varepsilon, \$ \rightarrow \varepsilon$
- $0, \varepsilon \rightarrow 0$
- $1,0 \rightarrow \varepsilon$

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.
If 'a' is read and 'b' is on top of stack, pop 'b'.
If 'a' is read, push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

Stack: $0$ $\$ $s = 0011$
Pushdown Automata (PDA)

\[ \varepsilon, \varepsilon \rightarrow \$ \]
\[ 1,0 \rightarrow \varepsilon \]
\[ \varepsilon, \$ \rightarrow \varepsilon \]

\[ a, b \rightarrow c \] If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

\[ \varepsilon, b \rightarrow c \] Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

\[ a, \varepsilon \rightarrow c \] If ‘a’ is read, push ‘c’.

\[ a, b \rightarrow \varepsilon \] If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack:

\[ s = 0011 \]
Pushdown Automata (PDA)

- $\varepsilon, \varepsilon \rightarrow \$ 
- $1,0 \rightarrow \varepsilon$ 
- $\varepsilon,\$ \rightarrow \varepsilon$

- $0, \varepsilon \rightarrow 0$ 
- $1,0 \rightarrow \varepsilon$

- $a, b \rightarrow c$ If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- $\varepsilon, b \rightarrow c$ Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- $a, \varepsilon \rightarrow c$ If ‘a’ is read, push ‘c’.
- $a, b \rightarrow \varepsilon$ If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack: $s = 0011$
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $s = 0011$
Pushdown Automata (PDA)

\[ a, b \rightarrow c \] If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

\[ \varepsilon, b \rightarrow c \] Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

\[ a, \varepsilon \rightarrow c \] If ‘a’ is read, push ‘c’.

\[ a, b \rightarrow \varepsilon \] If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack:

\[ s = 0011 \]
Pushdown Automata (PDA)

If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

If ‘a’ is read, push ‘c’.

If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack: $s = 0011$
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.
Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.
If 'a' is read, push 'c'.
If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $s = 00111$
Pushdown Automata (PDA)

- \( \epsilon, \epsilon \rightarrow \$$
- \( 1,0 \rightarrow \epsilon \)
- \( \epsilon, \$$ \rightarrow \epsilon \)

If \( a \) is read from input and \( b \) is on top of stack, pop \( b \) and push \( c \).

Without reading input, if \( b \) is on top of stack, pop \( b \) and push \( c \).

If \( a \) is read, push \( c \).

If \( a \) is read and \( b \) is on top of stack, pop \( b \).

Stack: \( s = 00111 \)
The given Pushdown Automata (PDA) has the following rules and transitions:

- $\varepsilon, \varepsilon \rightarrow \$$
- $1, 0 \rightarrow \varepsilon$
- $\varepsilon, \$ \rightarrow \varepsilon$
- $a, b \rightarrow c$: If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.
- $\varepsilon, b \rightarrow c$: Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.
- $a, \varepsilon \rightarrow c$: If 'a' is read, push 'c'.
- $a, b \rightarrow \varepsilon$: If 'a' is read and 'b' is on top of stack, pop 'b'.

The initial stack configuration is $s = 00111$. The diagram shows the transitions between states.
Pushdown Automata (PDA)

If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.

If ‘a’ is read, push ‘c’.

If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack: $\epsilon, \epsilon \rightarrow \$

1,0 \rightarrow \epsilon

\epsilon, \$ \rightarrow \epsilon

0, \epsilon \rightarrow 0

a, b \rightarrow c

\epsilon, b \rightarrow c

a, \epsilon \rightarrow c

a, b \rightarrow \epsilon

s = 00111
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $s = 00111$
If ‘$a$’ is read from input and ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.

Without reading input, if ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.

If ‘$a$’ is read, push ‘$c$’.

If ‘$a$’ is read and ‘$b$’ is on top of stack, pop ‘$b$’.

Stack: \[ s = \text{00111} \]
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.
If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

The stack configuration is $S = 0011$. The diagram shows the transitions in a Pushdown Automata (PDA).
Pushdown Automata (PDA)

- \(\varepsilon, \varepsilon \rightarrow \$\)
- \(1,0 \rightarrow \varepsilon\)
- \(\varepsilon, \$ \rightarrow \varepsilon\)
- \(0, \varepsilon \rightarrow 0\)
- \(1,0 \rightarrow \varepsilon\)

\[s = 001\]

\(a, b \rightarrow c\) If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

\(\varepsilon, b \rightarrow c\) Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

\(a, \varepsilon \rightarrow c\) If 'a' is read, push 'c'.

\(a, b \rightarrow \varepsilon\) If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack:

\(s = 001\)
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack: $001$

$s = 001$
Pushdown Automata (PDA)

- $\varepsilon, \varepsilon \rightarrow \$$
- $1,0 \rightarrow \varepsilon$
- $\varepsilon,\$$ \rightarrow \varepsilon$

- $a,b \rightarrow c$
  If ‘$a$’ is read from input and ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.

- $\varepsilon, b \rightarrow c$
  Without reading input, if ‘$b$’ is on top of stack, pop ‘$b$’ and push ‘$c$’.

- $a, \varepsilon \rightarrow c$
  If ‘$a$’ is read, push ‘$c$’.

- $a,b \rightarrow \varepsilon$
  If ‘$a$’ is read and ‘$b$’ is on top of stack, pop ‘$b$’.

$s = 001$
Pushdown Automata (PDA)

\[ \varepsilon, \varepsilon \rightarrow \$ \]
\[ 0, \varepsilon \rightarrow 0 \]
\[ 1,0 \rightarrow \varepsilon \]
\[ \varepsilon, \$, \varepsilon \rightarrow \varepsilon \]

- \( a, b \rightarrow c \) If ‘a’ is read from input and ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- \( \varepsilon, b \rightarrow c \) Without reading input, if ‘b’ is on top of stack, pop ‘b’ and push ‘c’.
- \( a, \varepsilon \rightarrow c \) If ‘a’ is read, push ‘c’.
- \( a, b \rightarrow \varepsilon \) If ‘a’ is read and ‘b’ is on top of stack, pop ‘b’.

Stack:

\[ s = 001 \]
Pushdown Automata (PDA)

If 'a' is read from input and 'b' is on top of stack, pop 'b' and push 'c'.

Without reading input, if 'b' is on top of stack, pop 'b' and push 'c'.

If 'a' is read, push 'c'.

If 'a' is read and 'b' is on top of stack, pop 'b'.

Stack:

\[s = 001\]
PDA Example

Build PDA to recognize:

\{ωω^R: ω \in \{0,1\}^* \text{ and } ω^R = ω \text{ written backwards}\}
PDA Example

Build PDA to recognize:

\( \{ \omega \omega^R : \omega \in \{0,1\}^* \text{ and } \omega^R = \omega \text{ written backwards} \} \)
CFL Pumping Lemma

Pumping Lemma: Given a context-free language $L$, $\exists$ a number $p$ such that any string $s \in L$, with $|s| \geq p$, can be divided into five pieces, $s = uvxyz$ satisfying:

1. $uv^ixy^iz \in L, \forall i \geq 0$.
2. $|vy| > 0$.
3. $|vxy| \leq p$. 
CFL Pumping Lemma

Pumping Lemma: Given a context-free language $L$, $\exists$ a number $p$ such that any string $s \in L$, with $|s| \geq p$, can be divided into five pieces, $s = uvxyz$ satisfying:

1. $uv^ixy^iz \in L, \forall i \geq 0$.
2. $|vy| > 0$.
3. $|vxy| \leq p$.

Claim: The language $L = \{0^n1^n0^n: n \geq 0\}$ is not context-free.

Proof: Suppose it is. Let $p$ be from the pumping lemma.

Let $s = 0^p1^p0^p$. ($s \in L$ and $|s| \geq p$)

$|vxy| \leq p \Rightarrow v$ and $y$ cannot contain 0’s from the front and back. So, pumping $v$ and $y$ up will lead to uneven 0’s.
Language Hierarchy

\[ \{0^n1^n0^n : n \geq 0\} \]