1. Give a regular expression that will accept all valid names. A name consists of a first name, an optional middle name or initial, and a last name, separated by spaces. First, middle and last names start with a capital letter and are followed by zero or more lowercase letters. If the name has a middle initial, instead of a middle name, it must be a capital letter followed by a period. Examples of valid names include:

- Joe Public
- Joe Q. Public
- Joe Quincy Public

Examples of invalid names include:

- Joe P. (no last name)
- Joe Q Public (middle initial missing a period)
- Joe Qu. Public (middle initial more than one letter)
- Joe Quincy Reginald Public (two middle names)
- Joe Quincy Public, the Third (more than just three names)

Assume that Σ (the alphabet) for the strings you are accepting is all capital letters, all lowercase letters, and ‘.’

**Answer:** Assume that the character ‘.’ represents a space. Define the following character class (which captures any single name):

\[ N = [A-Z][a-z]^{*} \]

We can then define the language using the following regular expression:

\[ N \cdot (N[A-Z])^{*} \cdot N \]

2. Give a DFA for that regular expression.

**Answer:**
3. Give a non-deterministic FSA for the following regular expression:

\((ab^*c)(a^+bc^*)\)

**Answer:** Here is one possible solution, based on building \(ab^*c\) and \(a^+bc^*\) first.

4. Produce the deterministic equivalent of the NFA you built in question 3. Show both the graphical representation and the tabular representation.
Answer: Let us proceed by the subset construction

<table>
<thead>
<tr>
<th>State(s)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Final?</th>
<th>New state</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, 5}</td>
<td>{3, 6}</td>
<td>err</td>
<td>err</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>{3, 6}</td>
<td>{6}</td>
<td>{3, 7, 8}</td>
<td>{4, 8}</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>{6}</td>
<td>{6}</td>
<td>{7, 8}</td>
<td>err</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>{3, 7, 8}</td>
<td>err</td>
<td>{3}</td>
<td>{4, 7, 8}</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>{4, 8}</td>
<td>err</td>
<td>err</td>
<td>err</td>
<td>Yes</td>
<td>5</td>
</tr>
<tr>
<td>7, 8</td>
<td>err</td>
<td>err</td>
<td>{7, 8}</td>
<td>Yes</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>err</td>
<td>{3}</td>
<td>{4, 8}</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>4, 7, 8</td>
<td>err</td>
<td>err</td>
<td>{7, 8}</td>
<td>Yes</td>
<td>8</td>
</tr>
</tbody>
</table>

The graphical representation is given below:

5. Minimize the deterministic FSA.

The only two states that behave the same way are 6 and 8, so they can be merged.
Note that it doesn’t matter that 3 transitions to 6 on b, but not 8. Once you’re in 6 or 8, the two states behave the same.

6. Can the language \((i \, g)^i, i \geq 0\) be recognized by an FSA? Why or why not?
   No, it cannot. We need to somehow determine how many ‘(‘s are seen before matching the same number of ‘)’s. This requires a potentially unbounded number of states.

7. Can the language \((k \, g)^k\) for one particular \(k\) be recognized by an FSA? Why or why not?
   Yes, it can. We can use \(k\) states to recognize the ‘(‘s, one state to recognize the ‘g’, and \(k\) more states to recognize the ‘)’s.