PLEASE NOTE: We will skip Section 3.2.1. Opting instead for the state elimination method of 3.2.2. Because of time, we will have to skip Section 3.3 as well, but some of this was covered in class already.

We will also only tread lightly on Section 3.4. Instead, here is a nonexhaustive summary of the useful algebraic identities for regular expressions/languages. Here, we won’t distinguish between a regular expression and the language it denotes. You can use these freely to simplify regular expressions and languages. \( R, S, \) and \( T \) are arbitrary languages (expressions).

1. Identities involving +:
   
   (a) \( R + S = S + R \). (The + operation is commutative.)
   
   (b) \( (R + S) + T = R + (S + T) \). (The + operation is associative.)
   
   (c) \( R + \emptyset = R \). (\( \emptyset \) is the identity under +.)
   
   (d) \( R + R = R \). (The + operation is idempotent.)
   
   (e) \( R? = R + \epsilon \). (Definition of \( R? \).)

2. Identities involving concatenation:

   (a) \( (RS)T = R(ST) \). (Concatenation is associative.)
   
   (b) \( R\epsilon = \epsilon R = R \). (\( \epsilon \) is the identity under concatenation.)
   
   (c) \( \emptyset R = R\emptyset = \emptyset \). (\( \emptyset \) is an annihilator under concatenation.)
   
   (d) \( R(S + T) = RS + RT \). (Left distributive law.)
   
   (e) \( (S + T)R = SR + TR \). (Right distribution law.)

3. Identities involving star (Kleene closure):

   (a) \( \emptyset^* = \epsilon^* = \epsilon \).
   
   (b) \( (R^*)^* = R^* \). (No need for two stars in a row.)
   
   (c) \( R^+ = RR^* = R^* R \). (Definition of \( R^+ \).)
   
   (d) \( R^* + \epsilon = R^* + \epsilon = R^* \).
   
   (e) \( (R^+)^* = (R^*)^+ = (R^?)^* = (R^*) = R^* \).
   
   (f) \( (R + \epsilon)^* = R^* \).
   
   (g) \( (R^* S^*)^* = (R + S)^* \).
   
   (h) \( R^* R^* = R^* \).
   
   (i) \( R(SR)^* = (RS)^* R \).
Example: Use just the identities to show that \( r^* + r = r^* \). Answer:

<table>
<thead>
<tr>
<th>Equation</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r^* + r )</td>
<td>( (r^* + \epsilon) + r )</td>
</tr>
<tr>
<td>= ( r + (r^* + \epsilon) )</td>
<td>1(a)</td>
</tr>
<tr>
<td>= ( r(r^* + \epsilon) + \epsilon )</td>
<td>1(b)</td>
</tr>
<tr>
<td>= ( (r + rr^*) + \epsilon )</td>
<td>3(c)</td>
</tr>
<tr>
<td>= ( (r\epsilon + rr^*) + \epsilon )</td>
<td>2(b)</td>
</tr>
<tr>
<td>= ( r(\epsilon + r^*) + \epsilon )</td>
<td>2(d)</td>
</tr>
<tr>
<td>= ( rr^* + \epsilon )</td>
<td>1(a) and 3(d)</td>
</tr>
<tr>
<td>= ( r^* + \epsilon )</td>
<td>3(c)</td>
</tr>
<tr>
<td>= ( r^* )</td>
<td>3(d)</td>
</tr>
</tbody>
</table>

1. (Optional) Using just the identities 1(a)–3(i) above, derive the following identities, as in the example above.

   (a) \( a^* + aa = a^* \).
   (b) \( a(ba)^*b = (ab)^+ \).
   (c) \( (a + b)^* + a = (a + b)^* \).

2. Do Exercise 3.2.3.
3. Do Exercise 3.2.4(b,c).

4. Recall the DFA \( D \) we constructed that accepts a binary string iff it has an odd number of 1’s:

\[
\begin{array}{c|cc}
\rightarrow & 0 & 1 \\
\hline 
A & A & B \\
B & B & A \\
\end{array}
\]

   (a) Convert \( D \) into an equivalent clean \( \epsilon \)-NFA using the clean-up procedure in class (add a new start state, a new final state, and some \( \epsilon \)-transitions).
   (b) Use the state elimination method to convert \( D \) to a regular expression. Eliminate state \( A \) first, then \( B \).

5. Same exercise as before, except make \( A \) the final state (so that \( D \) accepts a string iff it has an even number of 1’s).

6. (Optional) Recall the “product” DFA \( P \) that counts an even number of zeros and an odd number of ones:

\[
\begin{array}{c|ccc}
\rightarrow & 0 & 1 \\
\hline 
EE & OE & EO \\
OE & EE & OO \\
EO & OO & EE \\
OO & EO & OE \\
\end{array}
\]

   Use the state elimination method to convert \( P \) to a regular expression. (To control the complexity, you may wish to define names for intermediate regexps.)