1. Apply the algorithm of Section 3.2, as illustrated in Figs. 3.16 and 3.17, without deviating or simplifying.

I can’t draw (here, conveniently), but the result has the following structure:

(a) Fig. 3.17(a) separates the whole RE into two \( \Lambda \)-NFAs for \((ab)^*c\) \((R)\) and \(bc^*\) \((S)\).
(b) Fig. 3.17(b) separates the first of these into \( \Lambda \)-NFAs for \((ab)^*\) \((R)\) and \(c\) \((S)\).
(c) Fig. 3.17(c) transforms the first of these into a single \( \Lambda \)-NFA with \(ab\) for \(R\).
(d) And so on.

2. Apply the subset construction algorithm of Section 2.3.5.

The resulting DFA has four states \(\{p\}, \{p, q\}, \{p, q, r, s\}\) and \(\{p, t\}\), each of which is a subset of the states of the initial NFA), initial state \(\{p\}\), final states \(\{p, q, r, s\}\) and \(\{p, t\}\). The transition table for the resulting DFA is as follows:

<table>
<thead>
<tr>
<th>(\rightarrow)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>({p})</td>
<td>({p, q})</td>
<td>({p})</td>
</tr>
<tr>
<td>({p, q})</td>
<td>({p, q, r, s})</td>
<td>({p, t})</td>
</tr>
<tr>
<td>(<em>{p, q, r, s})</em></td>
<td>({p, q, r, s})</td>
<td>({p, t})</td>
</tr>
<tr>
<td>(<em>{p, t})</em></td>
<td>({p, q})</td>
<td>({p})</td>
</tr>
</tbody>
</table>

It’s important in these kinds of problems to only construct the states (subsets) of the DFA as required, and not to consider the set of all \(2^n\) possible subsets initially.

The initial NFA (and hence the final DFA) accepts the set of all strings of 0s and 1s ending with 00 or 01.

3. Apply the state elimination algorithm of Section 3.2.2.

If the states are eliminated in the order \(r, s, q\), the resulting regular expression is:

\[(1 + 0(0 + 10^*1)(1(0 + 10^*1))^*0)^*\]

Eliminating states in other orders will result in different, equivalent, regular expressions.

4. This is an exercise in the use of the pumping lemma for regular languages (Section 4.1). Proofs using the pumping lemma that languages are not regular always have the same general form.

Suppose the language \(L\) defined by this grammar is regular. Let \(n\) be the pumping lemma constant for \(L\). (Here comes the only creativity required in the proof; it’s very similar to that seen in previous examples.) Let \(w \in L\) be the string \((^n b)^n\), i.e., the expression with \(b\) between \(n\) pairs of nested parentheses. Then \(w\) can be expressed
as \(xyz\) with \(|xy| \leq n\) and \(y \neq \Lambda\). (Note that you have to consider all possible ways that \(w\) can be expressed as \(xyz\) satisfying these conditions.) As the first \(n\) symbols in \(w\) are all left parentheses, \(y\) in particular must consist only of left parentheses. By the pumping lemma, \(xz\) is also in \(L\). But \(xz\) has fewer left parentheses than right parentheses. As every string in \(L\) has the same number of left and right parentheses, \(xz\) cannot be in \(L\), which is a contradiction. Hence, \(L\) cannot be regular.

(Strictly speaking, we need an inductive proof that every string in \(L\) has the same number of left and right parentheses, but this proof is so trivial that if you can see the property you don’t need to give the proof.)

5. After I set this problem, I saw Exercise 5.1.8 in the text. This exercise gives the following elegant solution to the problem:

\[
S \rightarrow \Lambda \mid aSbS \mid bSaS
\]

Unfortunately, the proof that this grammar is correct is not trivial.

The solution that I had in mind was something like this:

\[
S \rightarrow \Lambda \mid aB \mid bA \\
A \rightarrow aS \mid bAA \\
B \rightarrow bS \mid aBB
\]

This also requires a nontrivial correctness proof. Both grammars clearly generate only strings with the desired property (in the second solution, \(A\) generates strings with one more \(a\) than \(bs\) and \(B\) similarly), but it’s not obvious, not at all obvious, that they generate all the strings with the desired property.

There may be other, simpler, solutions based on the same idea. Please let me know if you find one.