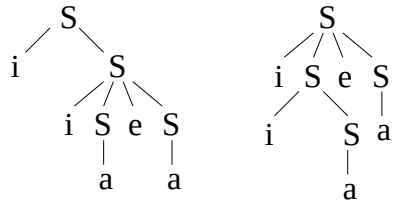
University of Nevada, Las Vegas Computer Science 456/656 Spring 2023

Answers to Assignment 2: Due Friday February 3, 2023

1. Prove that the grammar G given below is ambiguous by giving two different parse trees of the string `iiaea`. Both of those parse trees are correct, but only one of them is consistent with the usual rule for resolving ambiguity of if-then-else statements. Which one?

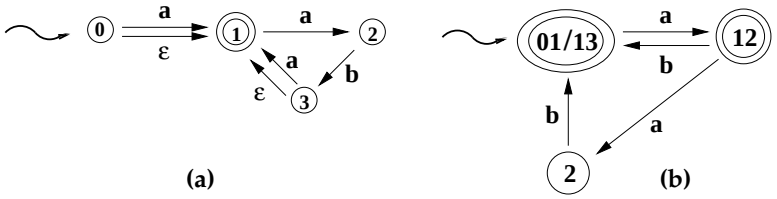
   1. `S` → `a`
   2. `S` → `iS`
   3. `S` → `iSeS`



   Using the usual rule of disambiguation, the tree on the left is the correct one.

2. Solve problems given in the handout `finiteAutomata.pdf` associated with the following figures.

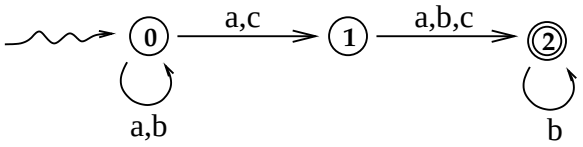
   (a) (Figure 7 of the handout.) Draw a minimal DFA equivalent to the NFA shown in (a). Your answer should be the same as (b).



	a	b
$\emptyset$	$\emptyset$	$\emptyset$
0	12	$\emptyset$
1	2	$\emptyset$
2	$\emptyset$	13
3	12	$\emptyset$
01	12	$\emptyset$
02	12	13
03	12	$\emptyset$
12	2	13
13	12	$\emptyset$
23	12	13
012	12	13
013	12	$\emptyset$
023	12	13
123	12	13
0123	12	13

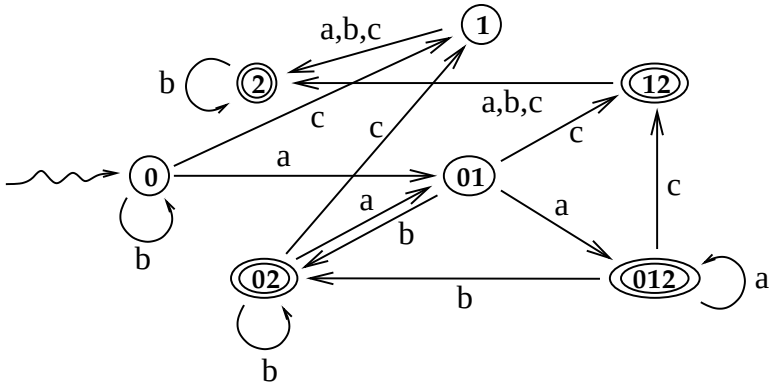
$Q = \{0, 1, 2, 3\}$, thus the states of the derived DFA are 2^Q, all subsets of Q. Here is the transition table of that DFA, where the the states of the DFA, all the subsets of Q are listed in canonical order. Of the 16 members of 2^Q, only five are reachable. The start state is 01. In (b) We do not show the dead state, the empty set. The only equivalence is that 01 and 13 are equivalent, hence the minimal DFA has four states including the dead state.

(b) (Figure 8 of the handout.) Give a minimal DFA equivalent to this NFA.

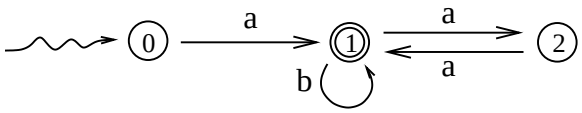


	a	b	c
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
0	01	0	1
1	2	2	2
2	$\emptyset$	2	$\emptyset$
01	012	2	12
02	01	02	1
12	2	2	2
012	012	02	12

We use the same technique we used for the previous problem. All 8 members of 2^Q are reachable, and there are no pairs of equivalent states of the DFA. We omit the dead state $\emptyset$ in the figure below.

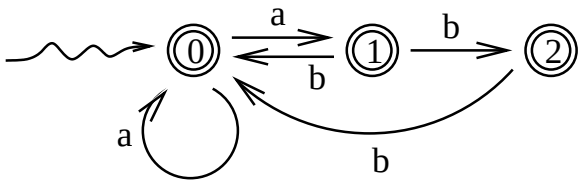


(c) (Figure 10 of the handout.) Write a regular expression for the language accepted by this NFA.

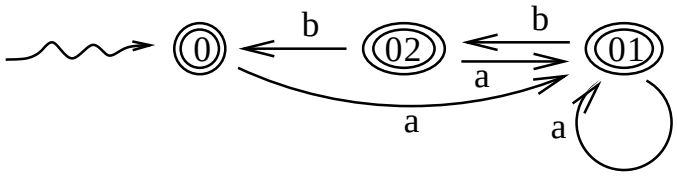


$$a(b + aa)^*$$

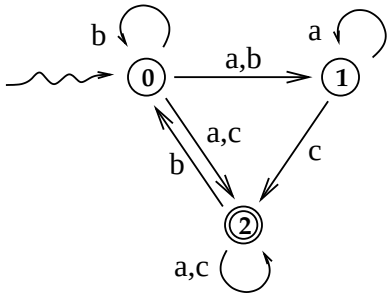
(d) (Figure 12 of the handout.) Give a minimal DFA equivalent to this NFA.



There are only four reachble states of the DFA, one of which is the dead state. No pair is equivalent.



(e) (Figure 13 of the handout.) Give a grammar for the language accepted by this NFA.



$$\begin{aligned} \texttt{S} &\rightarrow \texttt{bS} \mid \texttt{aA} \mid \texttt{bA} \mid \texttt{aB} \mid \texttt{cB} \\ \texttt{A} &\rightarrow \texttt{aA} \mid \texttt{cB} \\ \texttt{B} &\rightarrow \texttt{aB} \mid \texttt{cB} \mid \texttt{bS} \mid \lambda \end{aligned}$$

3. Write (do not prove) the pumping lemma for regular languages. (You will need to do this on the first examination, so try to understand it as you write it.)

   If L is a regular language there exists a number p, called a pumping length of L, such that, for any $w \in L$ of length at least p, there exist strings x, y, z such that the following four statemehnts hold:
   1. $w = xyz$
   2. $|xy| \leq p$ 3. $|y| \geq 1$ 4. For any integer $i \geq 0$, $xy^iz \in L$.

4. Please help me make a very important decision! As you know, the question of whether there is a $\mathcal{P}$–TIME algorithm which finds the factors of an integer is one of the most important unsolved problems in computation theory. In fact, the security of RSA encryption, which is used many times every day, depends on the hypothesis that no such algorithm exists.

   After many years of work, I have found an algorithm for the problem. Here is my code:

```
void primefactors(int n)
 // lists prime factors of n separated by commas
 {
  int f = 2;
  while (n%f != 0) f++;
   if(f == n) cout << n << endl; // n is prime
   else if(n%f == 0) // f is a prime factor of n
    {
     cout << f << ",";
     primefactors(n/f);
    }
 }
```

   You can easily see that the running time of the code is $O(n)$, which is certainly polynomial!

   I could either publish this result and become insanely famous, or keep it to myself (I'm sure you won't tell) and become insanely rich breaking RSA encryption for enormous fees. What should I do?

   An algorithm is in the class $\mathcal{P}$–TIME if its time complexity is bounded by a polynomial function of the number of bits of input. The number of bits of a the program is $\Omega(\log n)$. The time complexity of the program is $\Omega(n)$, which is an exponential function of the number of bits, since $n = 2^{\log_2 n}$. Realizing this, I canceled my world tour.