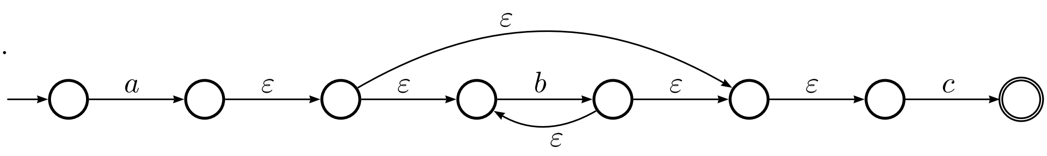
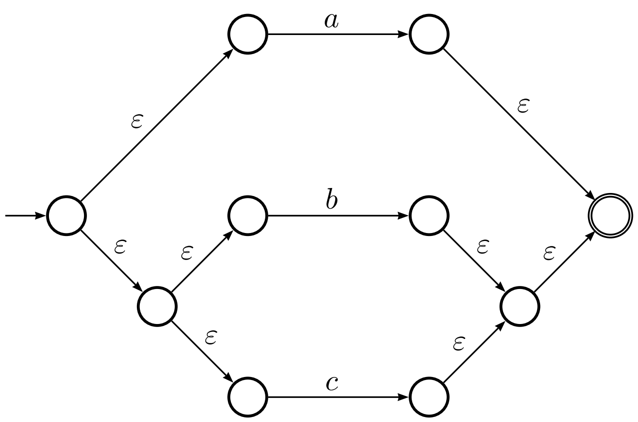
CS143 Problem Set 1 Solutions

Problem 1

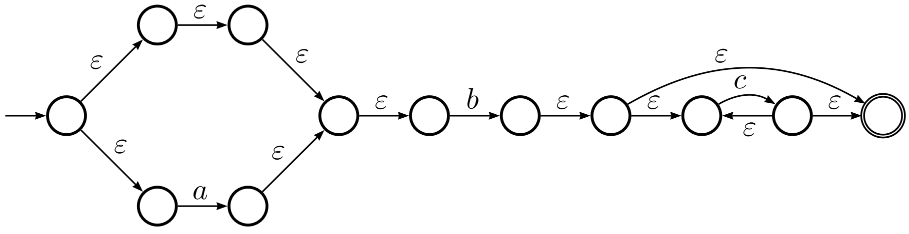
i.



ii.



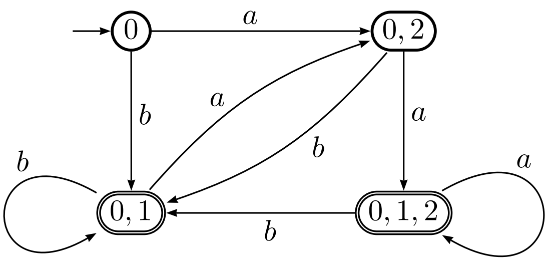
iii.



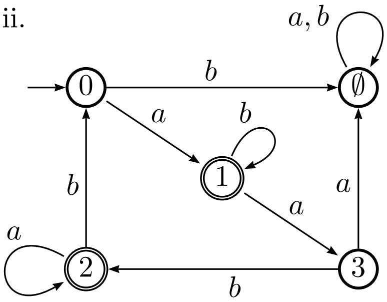
Problem 2

Let the states of the original NFAs be numbered top-to-bottom, left-to-right with the start state numbered 0. For example, in (i) the top row is numbered 0 and 1, while the bottom state is 2.

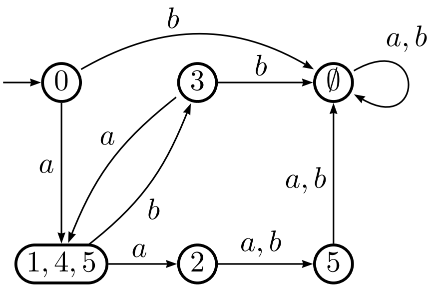
i.



ii.



iii.



Problem 3

i. The output is 132. The corresponding tokenization is “aaab”, “cc”, “abbb”.

ii. The output is 32a3. It is tokenized as “c”, “bbbb”, “a”, “c”. Since there is no rule to match “a”, flex echoes it back to the console.

iii. The output is 323. The tokenization is “c”, “bab”, “c”.

Problem 4

Consider the set of regular expressions

```
%%
aa
a
ab
```

and the input string "aab". This can be tokenized as "a" and "ab". Maximal-munch will instead tokenize it as "aa" and "b". Since "b" does not match any of the regular expressions, the algorithm will fail to tokenize this string.

Problem 5

i. There are a number of ways to solve this problem. Three possibilities are provided below.

Solution 1
Given a regular expression r, generate the NFA N using the construction discussed in class. Let N^R be the NFA that accepts the reverse of strings generated by R. Construct N^R from N by first reversing the direction of all the transitions. Let N's accept state be N^R's start state. Since the NFA construction produces only one accept state with no outgoing arrows, the start state of N^R has no incoming arrows. Similarly, let N's start state be N^R's accept state. As the start state had no incoming arrows, N^R's accept state has no outgoing arrows.

Claim: r generates a string w *iff* N^R accepts w^R.

Proof of Claim

- $\leftarrow$ Suppose r generates w. Then the NFA N for r accepts w along some path from the start state to the accept state. N^R by construction includes each step of this path in reverse (the start and accept states are inverted as are all the arrows). Since this is a valid path in N^R, w^R is accepted.
- $\rightarrow$ Suppose N^R accepts w^R. This corresponds to a path from N^R's start state to its accept state. But this same path is in N in reverse, so N must accept w and therefore r generates w.

Solution 2
Let r be a regular expression. Let r^R be the reverse regular expression of r. Define r^R recursively:

$REV(a)$

$REV(\varepsilon) = \varepsilon$

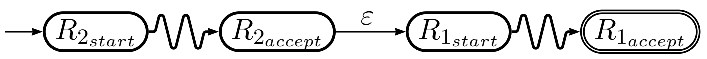
$REV(R1R2) = REV(R2)REV(R1)$

$REV(R1^*) = REV(R1)^*$

$REV(R1|R2) = REV(R1)|REV(R2)$

Construct the NFA from r^R using the standard construction.

Solution 3
Redefine the construction for R_1R_2 to be



ii. Consider the set of regular expressions

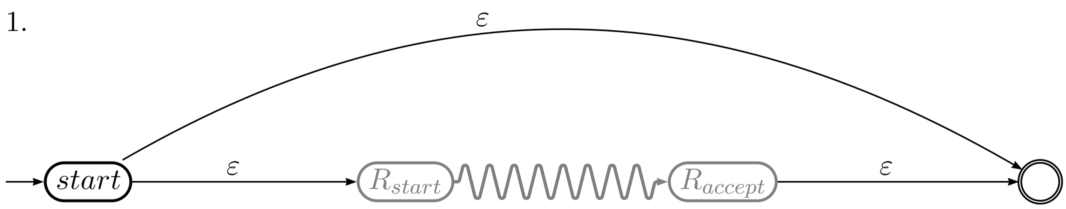
```
%%
a*
ab
b
```

and the input string "a^{n}b". Read forwards, it is tokenized as "a^n" and "b". In the reverse direction however, it is tokenized as "a^{n-1}" and "ab".

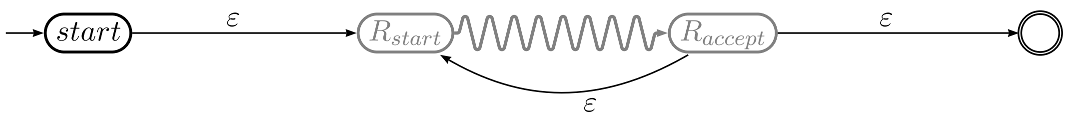
Problem 6

In the constructions below, let the dimmed states be the start and accept states of the NFA for R.

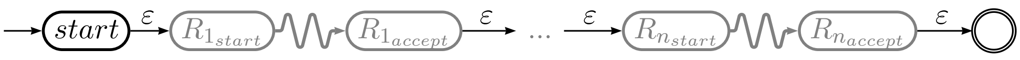
1.



2.



3. To construct R{n}, define a new start and accept state and concatenate R together n times using epsilon transitions. Notice for R{0} this will produce the NFA that accepts the empty string.



Problem 7

Consider the set of regular expressions

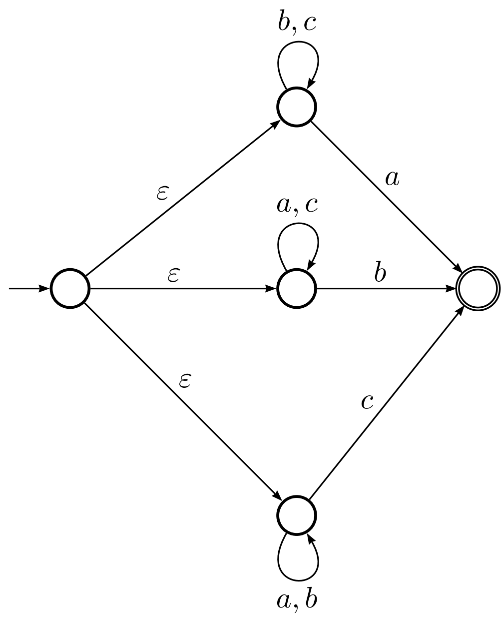
```
%%
a*b
a
```

and the function $f(n) = a^n$. The maximal-munch algorithm will scan all n characters of the string on the first pass before realizing it can only match the first "a". It will then scan the remaining $n-1$ characters in order to match the second "a", and so forth. In general on the i^{th} iteration, it will scan $n-i$ characters but only match one. Thus it spends $\sum^{n} n - i = \Theta(n^2)$ scanning.

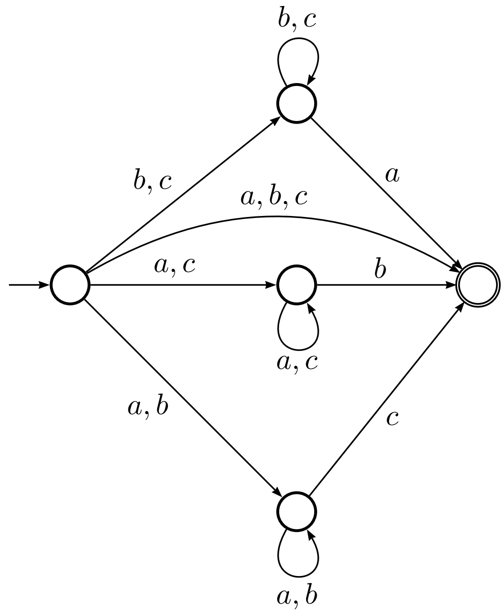
Problem 8

i. Two possible solutions are included below.

Solution 1



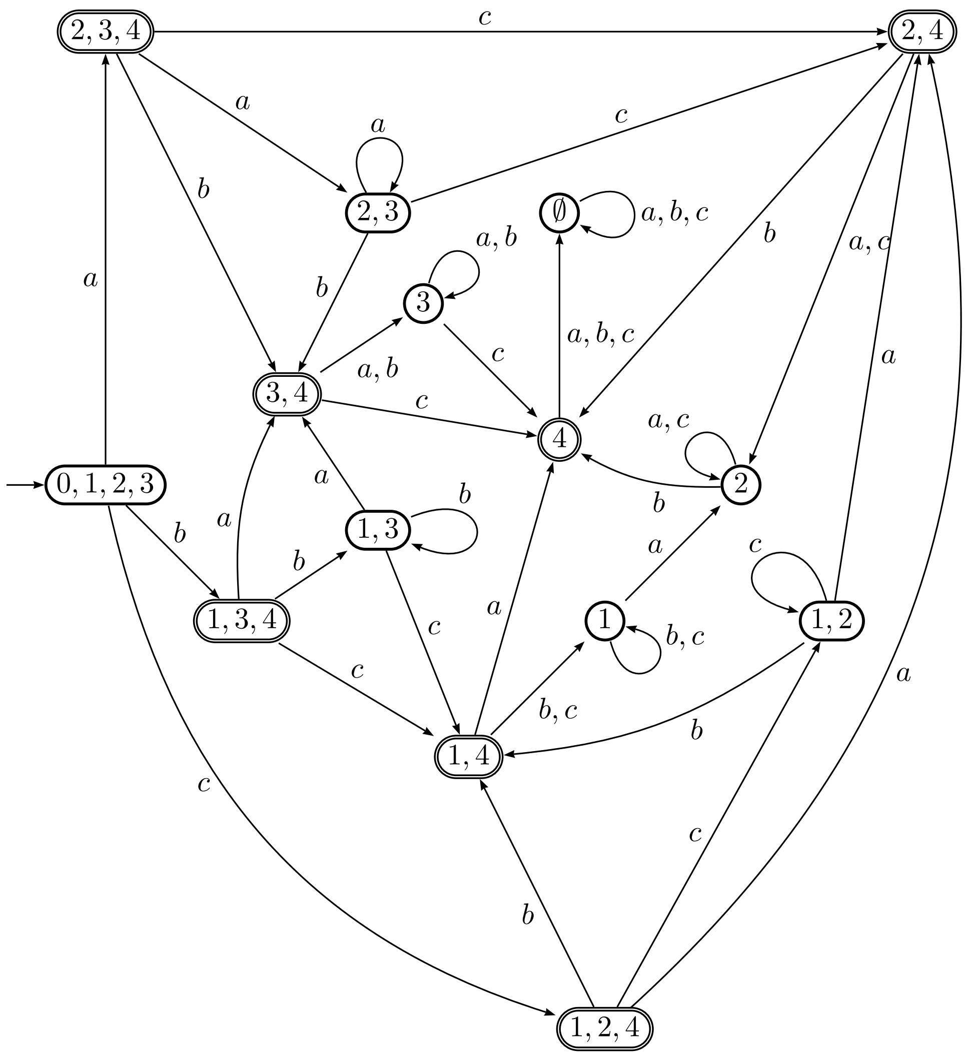
Solution 2



ii. There are 15 total states in the final DFA. Let the states be labeled left-to-right, top-to-bottom (e.g. 1, 2, and 3 are the middle column of states).

Below is the subset construction for Solution 1 from (i). It is displayed in two formats. The first is more visually clear whereas the second captures the intuitive structure of the DFA.

Option 1



Option 2