1 NFA and $\epsilon$-NFA Solutions

1. Every DFA is an NFA in which the transition relation only returns subsets of size at most 1; in other words, there is a unique arrow exiting every state for every symbol in the alphabet.

Can give any NFA where a state has two transitions for the same symbol, as an counter-example of the converse.

2. (a) We want to create an NFA for strings over $\{0, 1\}$ that end with “0001”. We can start with a machine (DFA) that only recognizes the string “0001”.

Recall in the DFA version of this question, we had to add multiple transitions for intermediary states. However, it is much simpler as an NFA. We only need to add transitions in state $\epsilon$ such that it can accept any number of 0 and 1 before entering the the next state:

(b) The NFA for this question looks very similar to the previous question. It will take 2 different paths instead of 1.

(c) We can start building the NFA by accepting string “1000”.
Now to accept any strings that starts with “1000”, we simply need another transition in the final state that goes back to itself on any input $\Sigma$.

![](image)

This is actually a DFA, since for every state in the diagram, there is at most one transition for each symbol. This is perfectly valid since all DFAs are also NFAs.

3. When we take the union of the two language $A \cup B$ using $\epsilon$-NFA. We create a new starting state, and have an $\epsilon$-transition going to the starting state of two NFAs.

![](image)

To concatenate two language, we add $\epsilon$-transitions between the final states of the machine for the first language to the starting state of the machine for the second. We also make all final states in the first machine no longer final. For language $C (A \cup B)$, we can apply the rules described above.

![](image)

There are multiple ways to simplify this $\epsilon$-NFA, we will leave it as an exercise.

4. There are two approaches to this problem. The first approach is quicker and more suitable for exams, basically drawing the DFA directly from the NFA. The second approach involves constructing a transition table, and is more suitable for computers.

To use subset construction, we want to determine the subsets that are necessary. We can start by renaming our states to something simple (say A, B, C, etc) and start the analysis with a table of transition relation.
The possible subsets we can get to are \{A,B\}, \{A,D\}, \{C\}, and \{E\}. The states C and E exist already, so we only have to construct two new states. For the transition relation, we take the union of the transition relations of each element in the subset. We repeat this process until we see no new subsets are created.

We look at the two tables again. State A can go to either AB or AD, and AB and AD don’t go to any 1-element subset. Therefore, we don’t have to draw the B or D states.

Any subset which contains a final state (C or E) will be final after the subset construction.