WEEK 8 WORKSHOP

MATH2301, SEMESTER 2, 2025

1. Warm-up: NFA Problems Level 0

1.1. **Problem.** When is an NFA said to *accept* a string? When is an NFA said to *reject* a string?

Solution. Accepts if at least one branch of the calculation goes through the entire string and ends on an accepting state. Rejects if either all branches die out, or every branch that goes through the entire string ends on a non-accepting state.

1.2. **Problem.** Let M be an NFA. Let $L = L(M)$ and $L^c = \Sigma^* - L$. If you take an NFA and switch all the accepting states to non-accepting states and vice-versa, will the language of the resulting machine be L^c? Why or why not?

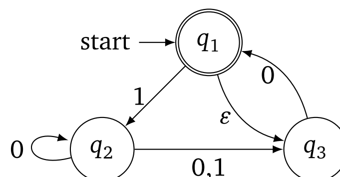
Solution. No, not usually! The new machine may still accept some strings that were accepted by the old machine. For example, look at the machine from Page 1 of the lecture on 03 October 2023. It accepts the string 010 via the sequence $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$. If you switched the accepting and non-accepting states, the resulting machine would still accept the string 010, via for example the sequence $q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0$.

1.3. **Problem.** What happens in the calculation of an NFA if from a given state, it is not possible to read the letter we are supposed to read?

Solution. We say that that branch "dies out", i.e., we do not continue the calculation for that branch.

2. NFA Problems Level 1

2.1. **Problem.** Consider the NFA shown below.



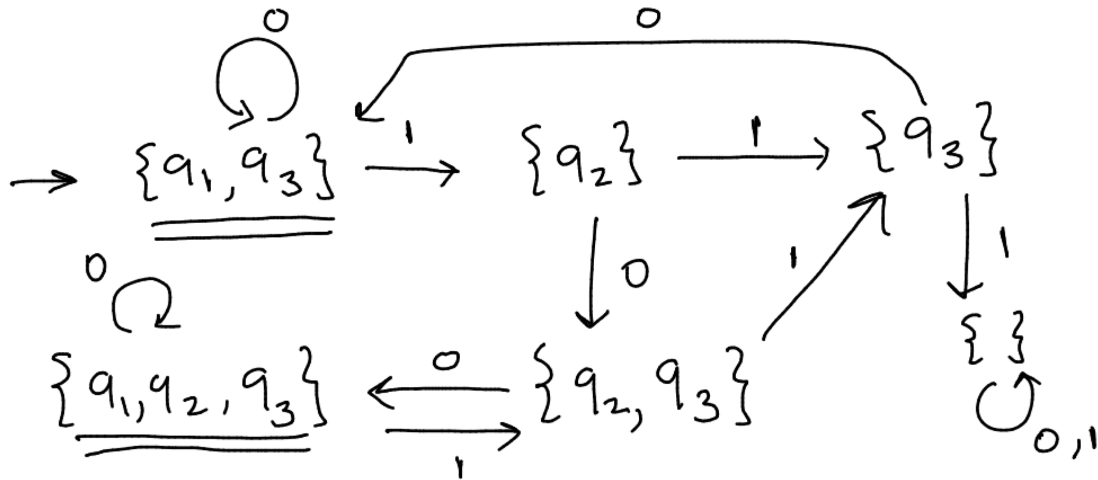
Find at least one string (ideally more!) that the NFA accepts and at least one that the NFA rejects.

Solution. The NFA accepts $0*, 100*$ (and many more) and rejects $11*$ (and many more).

2.2. **Problem.** Convert the NFA into an equivalent DFA.

Date: September 22, 2025.

Solution.



(I hope I did not make a mistake.)

3. NFA Problems Level 2

The next problems are about constructing machines that do certain things. Carefully convince each other why your solutions work. Some of these problems are hard, and take practice to solve. Don't get discouraged if you can't come up with solutions right away!

3.1. **Problem.** Let L be a language. Consider the following language.

$$L^{del} = \{xz \mid xyz \in L \text{ where } x, z \in \Sigma^*, y \in \Sigma\}.$$

Given an NFA that recognises L, construct an NFA that recognises L^{del}.

Hint: In your NFA, at any point once you've read some portion of the string, you should create the option to ignore one letter and then move on with the calculation.

Solution. We will take two copies of the original NFA; let's call them M_1 and M_2. We keep the transitions and will add new ones. The start state is the start state of M_1, and the accept states are just the accept states of M_2. Now we add extra transitions as follows. Consider any state $q_1 \in M_1$, any letter $y \in \Sigma$, and any state $q_2 \in \Delta(q, y)$. For any such choice, add an arrow labelled ε from q_1 to q_2. This simulates that we "deleted" the letter y, and then continued on in M_2 to read the rest of the string, accepting if we had accepted the original string.

3.2. **Problem.** Let L_1 and L_2 be languages. Let the *perfect shuffle* of L_1 and L_2 be the language

$$L = \{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1, \ldots, a_k, b_1, \ldots, b_k \in \Sigma \text{ and } a_1 \cdots a_k \in L_1 \text{ and } b_1 \cdots b_k \in L_2\}.$$

(The number k can be arbitrary). As a warm up to understand the construction:

(1) Take $L_1 = 0^*$ and $L_2 = 1^*$, and describe the perfect shuffle L.
(2) Take $L_1 = \{0, 00, 000\}$ and $L_2 = \{1, 11, 111\}$, describe the perfect shuffle L.

Now, given automata M_1 and M_2 recognising L_1 and L_2, respectively, construct an automaton M to recognise L. You may assume that M_1 and M_2 are deterministic, and construct a non-deteterministic M.

Here is a series of pointers to help you come to a complete solution of this problem.

(1) We'll have to somehow combine the DFAs recognising the two languages into a third one.
(2) We'll need to start at the start state of M_1, because we expect the first letter of any valid word to be a letter that's valid at the start of any word in L_1.
(3) Once we read a letter from L_1, we have to "pause" and read a letter from L_2 in order to be valid. But after that, we'll have to "resume" in M_1, which means we have to remember where we came from. Can you simulate this using a product-type construction?

Solution. Here is a solution sketch. The idea is that we start in M_1, then move to M_2, then back to M_1 and so on, each time remembering where we came from and where we are. We will use as our states two copies of the product of the states of M_1 and M_2. Let us call the first copy M_A and the second copy M_B, and our final machine will be a combination of M_A and M_B with several arrows. The elements of M_A will be written as (p,q) where p and q are states of M_1 and M_2 respectively. The elements of M_B will be written as (q,p) where q and p are states of M_2 and M_1 respectively.

The convention is that:

(1) If we're at state (p,q) in M_A, then we're *at* state p in the first machine but we *came from* the state q in the second machine.
(2) If we're at state (q,p) in M_B, then we're *at* state q in the second machine, but we *came from* the state p in the first machine.

To start with, we're at the state (p_1,q_1) in M_A (both start states). If we read a symbol $a \in \Sigma$, we transition to the state $(q_1, \delta_1(p_1,a))$ in M_B.

If we're at the state (q,p) in M_B, and we read a symbol $b \in \Sigma$, we transition to the state $(p, \delta_2(q,b))$ in M_A.

Finally, the accepting states are all in M_A, and have the form

$$\{(p,q) \mid q \text{ is an accepting state of } M_2\}.$$

Convince yourself that this solution works!

3.3. **Problem.** ($*$, bonus) If you're finished with the remainder of the worksheet, construct an automaton to recognise the *shuffle* of two regular languages L_1 and L_2, defined as follows:

$$L = \{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1, \ldots, a_k, b_1, \ldots, b_k \in \Sigma^* \text{ are such that } a_1 \cdots a_k \in L_1 \text{ and } b_1 \cdots b_k \in L_2\}.$$

Solution. Very similar to the previous, but with extra transitions, making it into an NFA. Here is a solution sketch. If we're in M_A, we can either jump to M_B right away, or after reading one valid letter of a word in L_1, or after reading several valid letters of a word in L_1. So we need to keep the original transition arrows, and add new ones as follows:

(1) From a state (p,q) in M_A, add an ϵ arrow to the state (q,p) in M_B.
(2) From a state (p,q) in M_A, and for any $a \in \Sigma$, add an arrow to $(\delta_1(p,a),q)$ in M_A. *This simulates that we can continue to read in L_1 for a little while before deciding to jump to reading in L_2.*
(3) Similarly from (q,p) in M_B, add an ϵ arrow to the state (p,q) in M_A.
(4) From a state (q,p) in M_B, and for any $b \in \Sigma$, add an arrow to $(\delta_2(q,b),p)$ in M_B

Convince yourself that this solution works!