Tutorial Sheet Solution

Table of contents

> ⚠ Warning
>
> **Important**: Your solution does not have to be identical (or even close) to mine, as long as you have a similar line of reasoning, it is more than acceptable.

Subjective Questions:

1. Define the following terms:
   a) Automaton
   b) Alphabet
   c) String
   d) Language
2. Given an alphabet $\Sigma = \{a, b\}$, list some valid strings over Σ.
3. Explain why we need a simpler way of discussing computing machines.
4. Describe the concept of an automaton and its role as a mathematical model of a computing device.

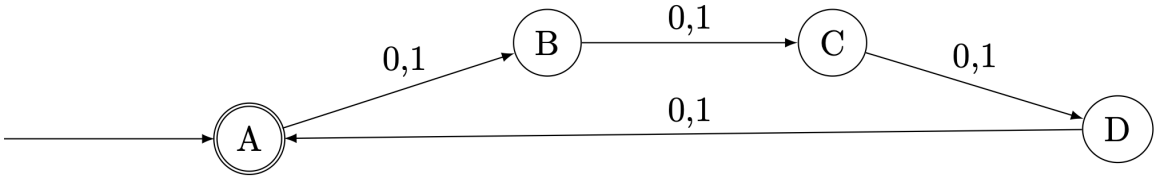
5. What are the reasons for building models in computing? Explain the significance of finite automata as abstractions of computers.
6. Define an alphabet and a string over an alphabet Σ. Provide examples.
7. Given an alphabet $\Sigma = \{a, b, c\}$, list some valid strings in the language of palindromes over Σ.

Objective Questions:

1. Define an NFA and explain how it differs from a DFA in terms of determinism.
2. Define the formal structure of a DFA and NFA (hint: look at the 5 tuple structure I disucssed in the lecture).

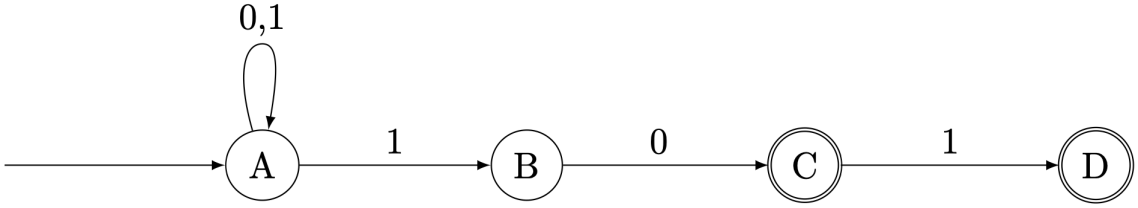
1 Design DFA

Q1. What language is accepted by the following DFA?



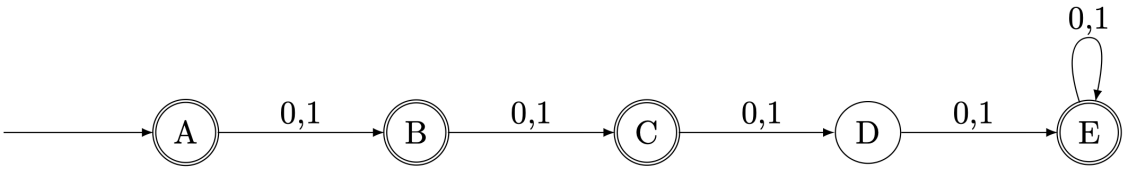
Solution: Binary strings of length 4 or multiples of 4, and the empty string.

Q2.What language is accepted by the following NFA?

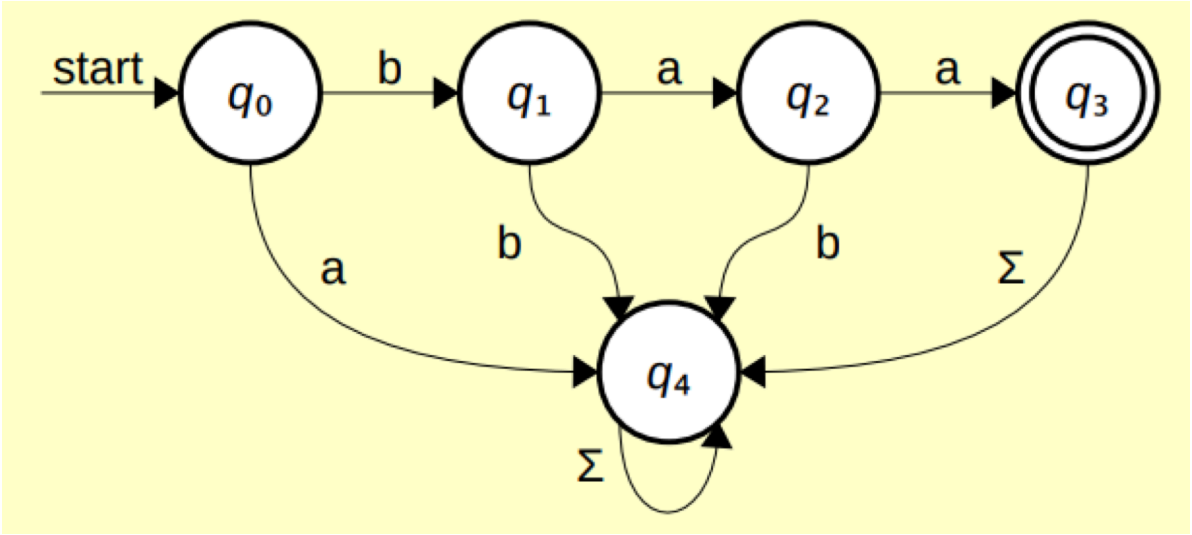


Solution: Binary strings ending in 10 or 101

Q3. Construct a DFA that accepts binary strings of any length, except 3.

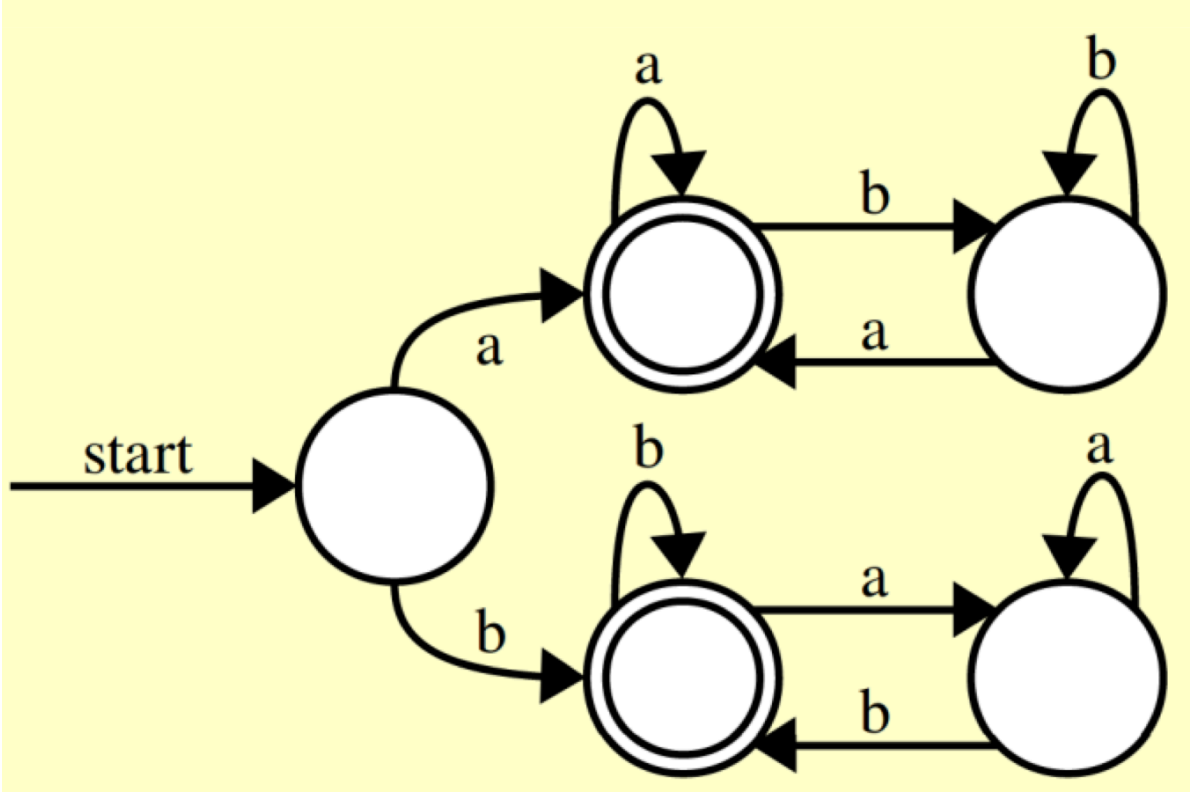


Q4. Let Σ = {a, b} and let L = {baa}. Design a DFA for L.



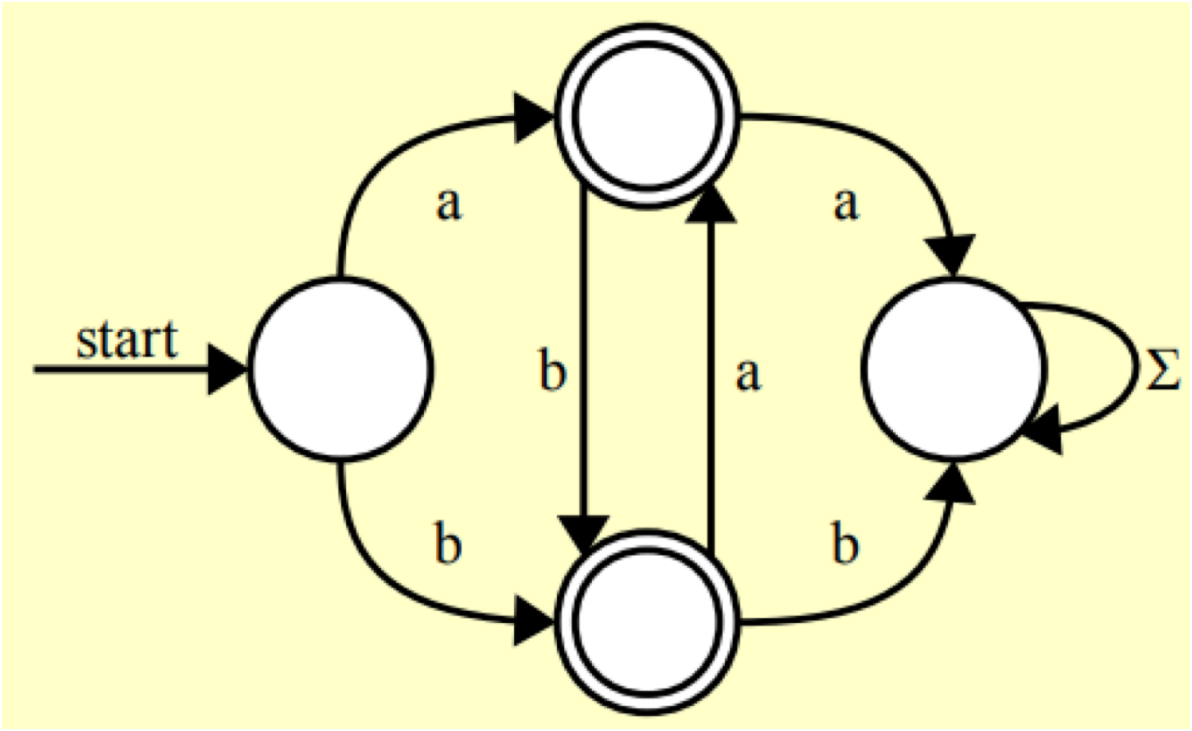
This basically consists of a chain of states leading up to baa with a dead state for any deviations from that string.

Q5. Let Σ = {a, b} and let L = { w Σ^* | w = and the first and last character of w are the same }. Design a DFA for L.



In the start state, we wait to see what the first character is. We then transition either to the top branch (first char- acter a) or the bottom branch (first character b). At that point, we just need to remember what the last character we read was and can use that to decide whether to accept.

Q6. Let $\Sigma = \{a, b\}$ and let $L = \{ w \quad \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Design a DFA whose language is L.



In the start state, we wait to see what the first character is. We then transition either to the top state (first charac- ter a) or the bottom state (first character b). The transitions then permit us to bounce back and forth between the two states as long as we alternate, and if we ever deviate from the rule we enter the dead state on the right.

Why we asked this question: There are a couple of canonical tricks in the design of DFAs. Part (i) was a check to make sure you remembered to include a transition on every state/symbol combination. Part (ii) demonstrates how to build an automaton that can remember one of finitely many pieces of information persistently (namely, what the first character is), and part (iii) combines the two.

2 Design NFA

2.1 Tip for Desiging NFA

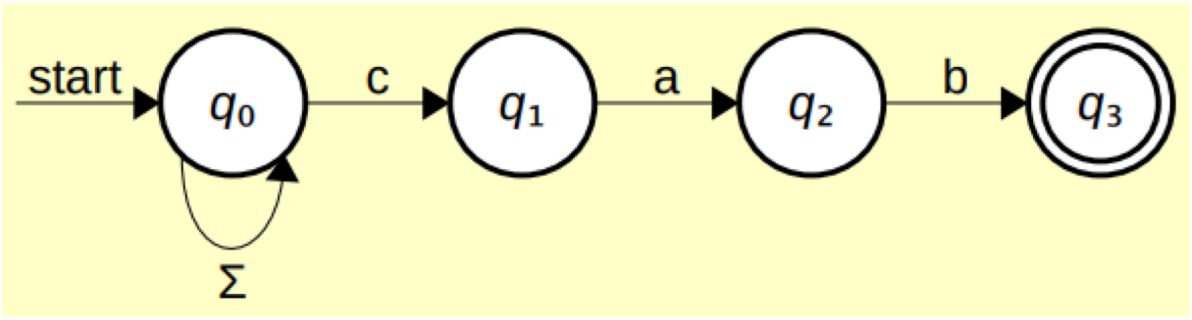
Designing a Non-deterministic Finite Automaton (NFA) can be a bit more flexible than designing a Deterministic Finite Automaton (DFA). Here are some tips to help you design an NFA:

1. **Understand the Language**: Before designing the NFA, make sure you understand the language or the set of strings you want the NFA to recognize.

2. **Start Simple**: Begin with a basic structure. If you're designing an NFA for a language that accepts strings ending in '01', start with that basic requirement.
3. **Use -Transitions**: One of the powers of NFAs is the ability to move from one state to another without consuming an input symbol, using -transitions. This can simplify your design in many cases.
4. **Multiple Transitions**: Remember that for a given state and input symbol, an NFA can have transitions to multiple states. Use this feature to represent multiple possibilities.
5. **Avoid Overcomplicating**: Just because you can have multiple transitions doesn't mean you always should. If a simpler design works, go with it.
6. **Design Incrementally**: If you're trying to design an NFA for a complex language, break the problem down. Design smaller NFAs for simpler languages and then combine them.
7. **Use Sub-automata**: For complex languages, design sub-automata for different parts of the language and then connect them using -transitions.
8. **Test with Strings**: Once you've designed your NFA, test it with various strings to ensure it accepts those it should and rejects those it shouldn't.
9. **Convert to DFA (Optional)**: If you find creating a NFA difficult, maybe you can solve the problem using a DFA, remember that any NFA can be converted to an equivalent DFA.

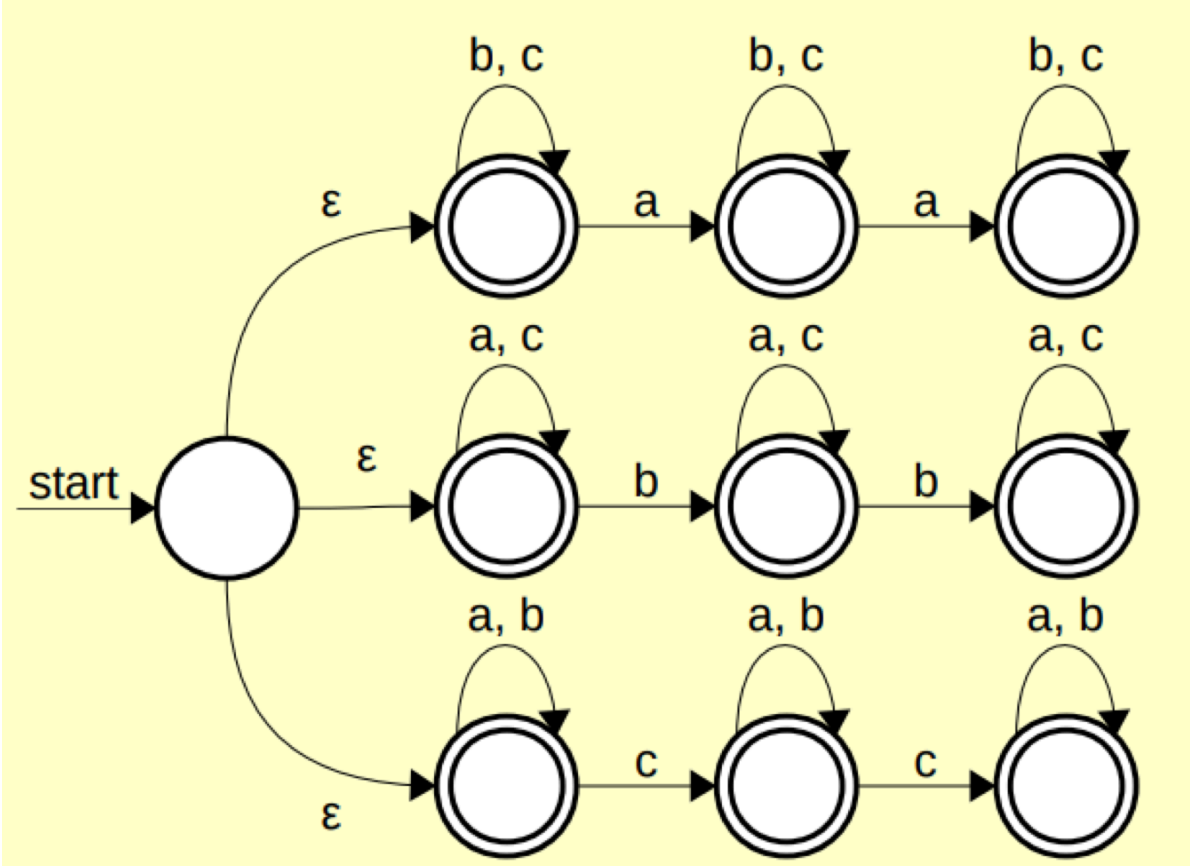
Remember, designing an NFA is as much an art as it is a science. With practice, you'll develop an intuition for creating efficient and effective NFAs.

Q7. Let $\Sigma = \{a, b, c\}$ and let $L = \{ w \quad \Sigma^* \mid w \text{ ends in cab} \}$. Design an NFA for L.



This NFA uses nondeterminism to guess when it's three characters from the end. If we guess correctly and the input does end in cab, we accept. The machine otherwise doesn't accept.

Q8. Let $\Sigma = \{a, b, c\}$ and let $L = \{ w \quad \Sigma^* \mid$ some character in Σ appears at most twice in w $\}$. Design an NFA for L.



Detailed instruction on designing a NFA for this question:

Design an NFA for the language (L) where (L = { w in Sigma^* | some character in Sigma appears at most twice in w }) and (Sigma = {a, b, c}).

The idea is to design an NFA that accepts strings where any character from (Sigma) appears at most twice. We'll design sub-automata for each character and then combine them.

This NFA uses nondeterminism to guess which character will appear at most twice. Once it does, it transitions into one of three different chains (one per possible character) that count the number of times that character appears. If we exceed the limit, the automaton dies off in that branch. This means that to accept, we have to guess the branch correctly, then survive long enough to accept.

Why I asked this question: Unlike DFA design, NFA design often focuses on determining what information you need to nondeterministically guess. This problem show off the hybrid model of using nondeterminism to guess some information, then determinism (in some form) to check it. Q8 was specifically designed to get you thinking about how to count with states.

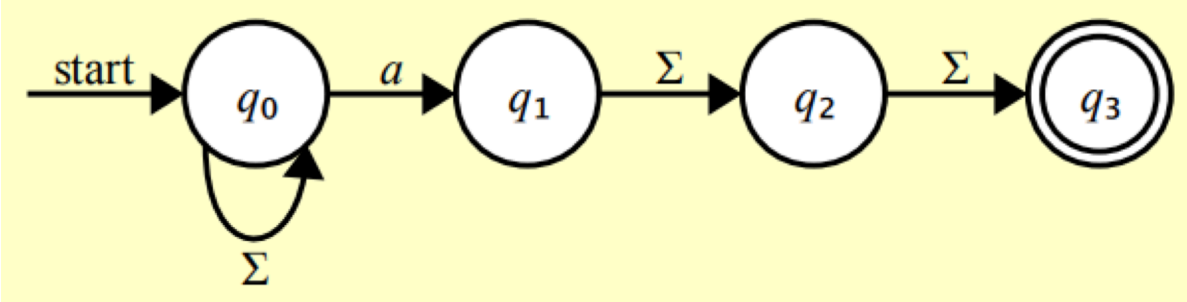
Here's a step-by-step design:

1. **NFA for 'a' appearing at most twice**:
    - Start state (q_0) transitions to state (q_1) on seeing 'a', (q_1) should be an accepting state as . Inputs such as bc, cc, b are also valid an we need to accept them, we do this by looping back on (q_0) when we see b or c.
    - State (q_1) transitions to state (q_2) (another accepting state) on seeing another 'a'. This should also be accepting state that loops back on b or c. The only thing we are focused on now is not accepting a string that has more than 2 a's.
    - State (q_2) is also accepting state but will transition to a dead state (q_d) on seeing a third 'a' alternatively you can also make the transition die at this state on seeing a.
2. **NFA for 'b' appearing at most twice**:
    - Similar to the NFA for 'a', but transitions are based on 'b'.
3. **NFA for 'c' appearing at most twice**:
    - Similar to the NFA for 'a', but transitions are based on 'c'.
4. **Combining the NFAs**:
    - When you try and combine them you would notice that each of the sub-automata work well but combining them creates new problems (what should be the combined start state). To resolve these issues, we create a new start state and use -transitions to move to one of these sub-automata (using the non-determinism to our benefit!)
5. **Other Transitions**:
    - From any state, on seeing a character not leading to a transition, stay in the same state. This ensures that other characters are effectively ignored while counting occurrences of 'a', 'b', and 'c'.

Look at the visual representation of the NFA above.

- Circles represent states.
- Arrows represent transitions with the character label.
- Double circles represent accepting states.
- I have not used the dead state as I let the transition die in my exmaple but you could definately use a dead state.

Q9. Let $\Sigma = \{a, b\}$ and let L = { w Σ^* | the third-from-last character of w is a }. Design an NFA for L. Your NFA should use at most four states.

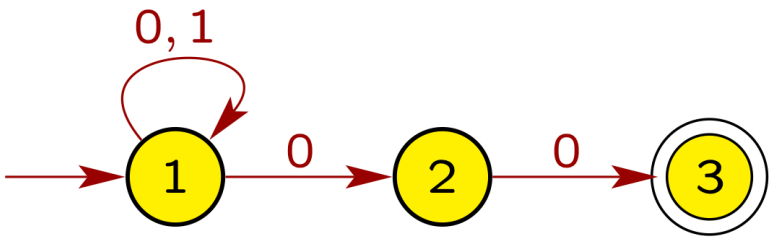


Along the lines of the "ends with cab" problem, this NFA stays in the start state until it nondeterministically guesses that it's three characters from the end:

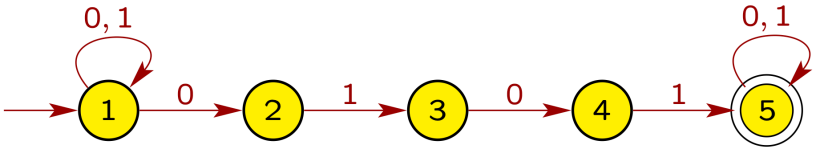
3 More Practice Questions

Q10. Give NFAs with the specified number of states recognizing each of the following languages. In all cases, the alphabet is $\Sigma = \{0, 1\}$.

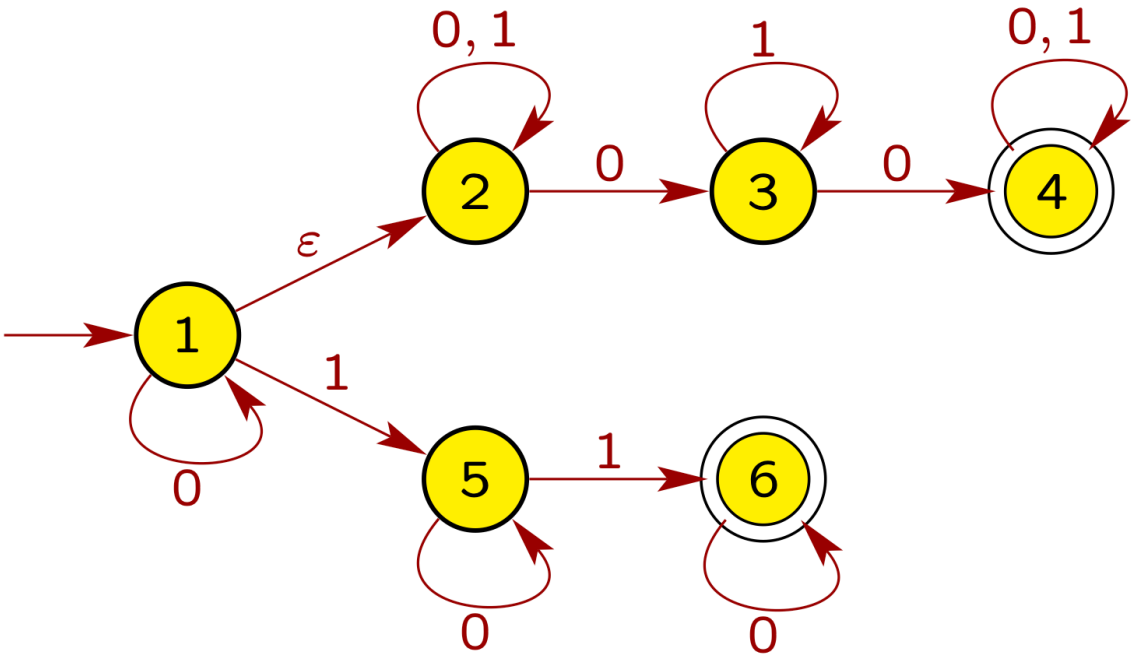
- Q10.1. The language { w $\in \Sigma*$ | w ends with 00 } with three states.



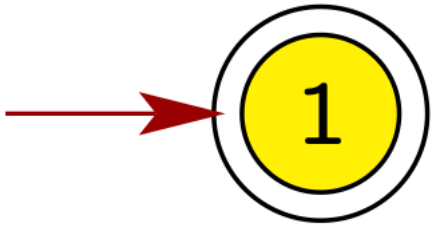
- Q10.2. The language { w $\in \Sigma*$ | w contains the substring 0101, i.e., w = x0101y for some x, y $\in \Sigma*$ } with five states.



- Q10.3. The language { w $\in \Sigma*$ | w contains at least two 0s, or exactly two 1s } with six states.



- Q10.4. The language { } with one state.



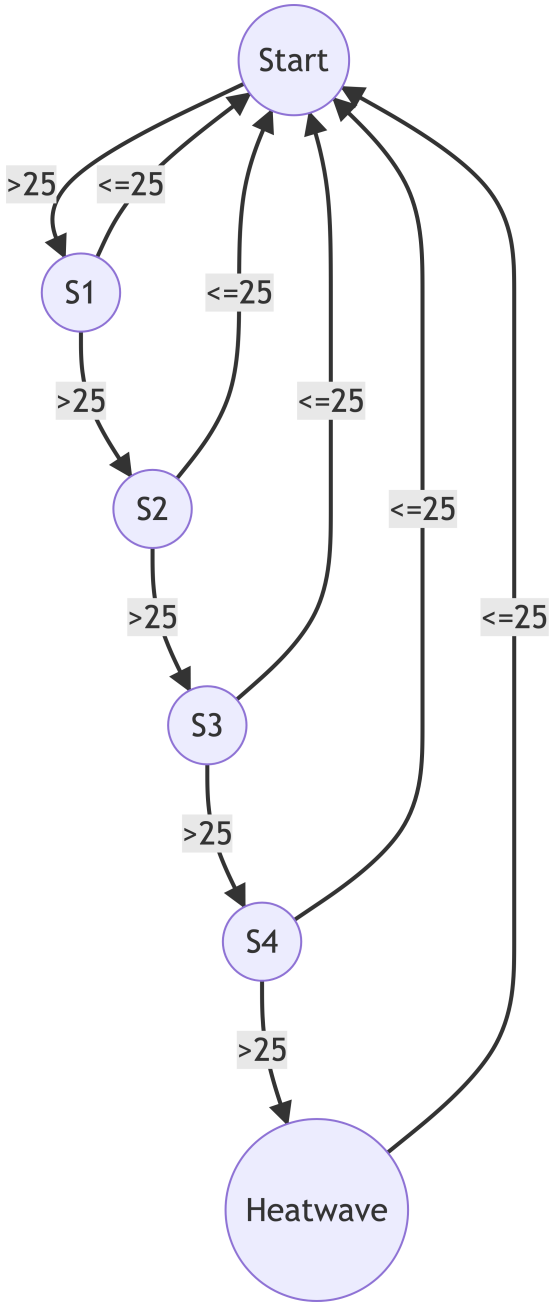
4 More simple/fun Questions

> ⚠ Warning
>
> **Important**: I will give you solutions to these questions but they may be slightly wrong, you need to figure out what is wrong and fix that [1].

1. **Heat Wave in The Netherlands**: Design a DFA to determine if a heat wave has occurred in The Netherlands (the definition of a heatwave is that the temperature should be greater than 25°C for 5 days straight).

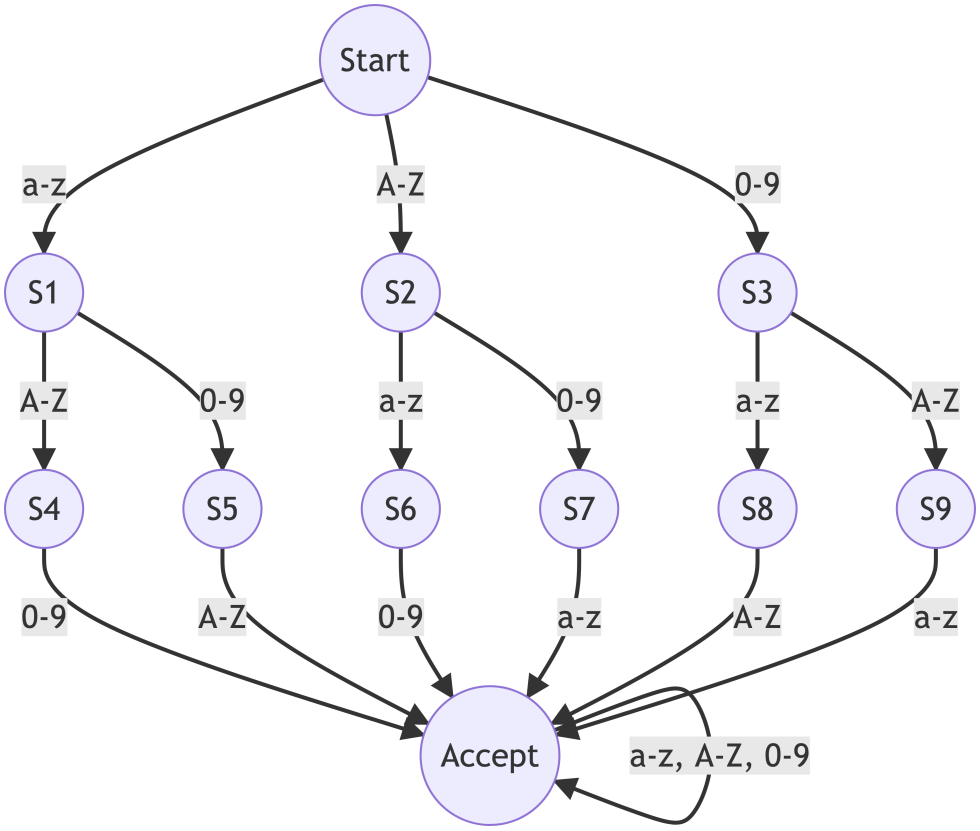
[1] Trust me this is better than having no solutions!

[L = { w in {<= 25, > 25}* | w { contains at least five consecutive } > 25 }]



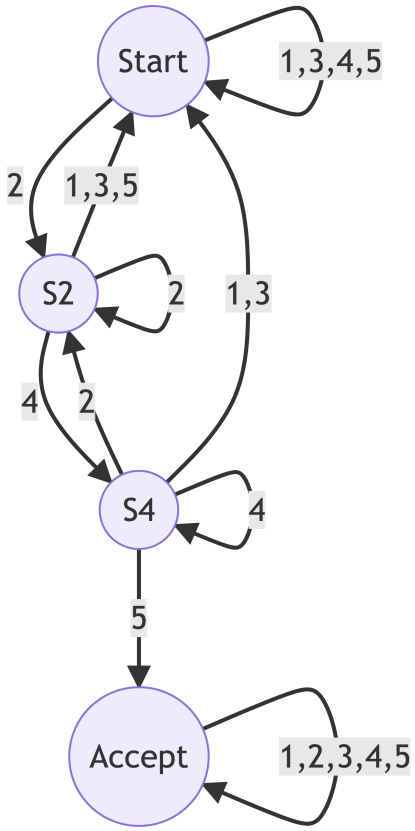
2. **Password Strength**: Design a DFA that accepts passwords that are at least 3 characters long, contain at least one uppercase letter, one lowercase letter, and one digit.

[L = { w in {a-z, A-Z, 0-9}* | w { contains at least one uppercase, one lowercase, one digit, and has length } >= 3 }]



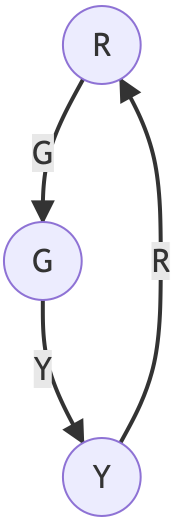
3. **Elevator Button Presses**: Design an NFA where the elevator goes to the 5th floor only if the 2nd and 4th-floor buttons are pressed consecutively.

[L = { w in {1,2,3,4,5}* | w { contains 2 followed by 4} }]



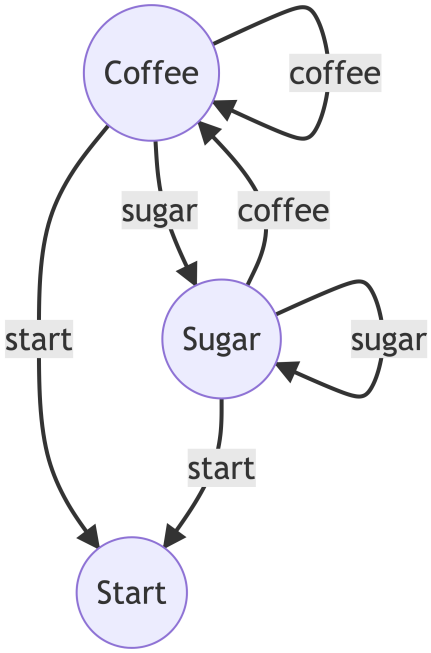
4. **Traffic Light Sequence**: Design a DFA that represents the sequence of traffic lights (Red -> Green -> Yellow -> Red).

[L = { w in {R, G, Y}* | w { follows the pattern } RGY }]



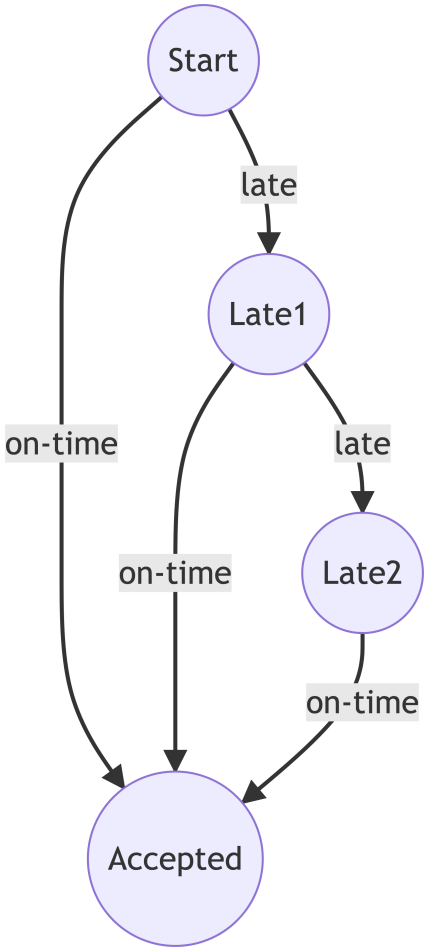
5. **Coffee Machine**: Design a DFA that accepts sequences where a user selects a coffee type, adds sugar (optional), and then presses the start button.

[L = { w in {coffee, sugar, start}* | w { starts with coffee and ends with start} }]



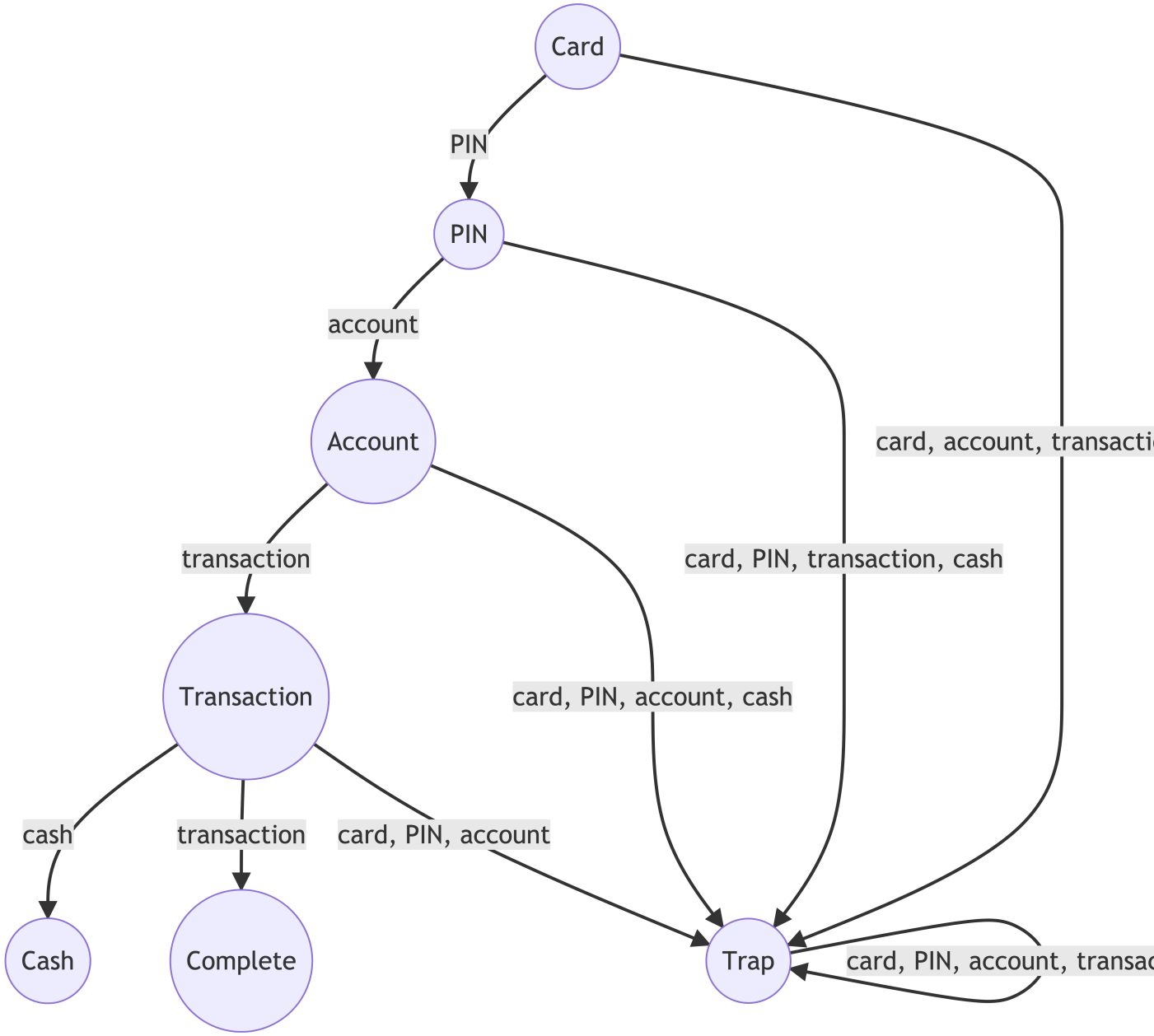
6. **Library Book Return**: Design an NFA that accepts if a book is returned either on or before its due date or within two days after its due date.

[L = { w in {on-time, late}* | w { is either on-time or late up to two days} }]



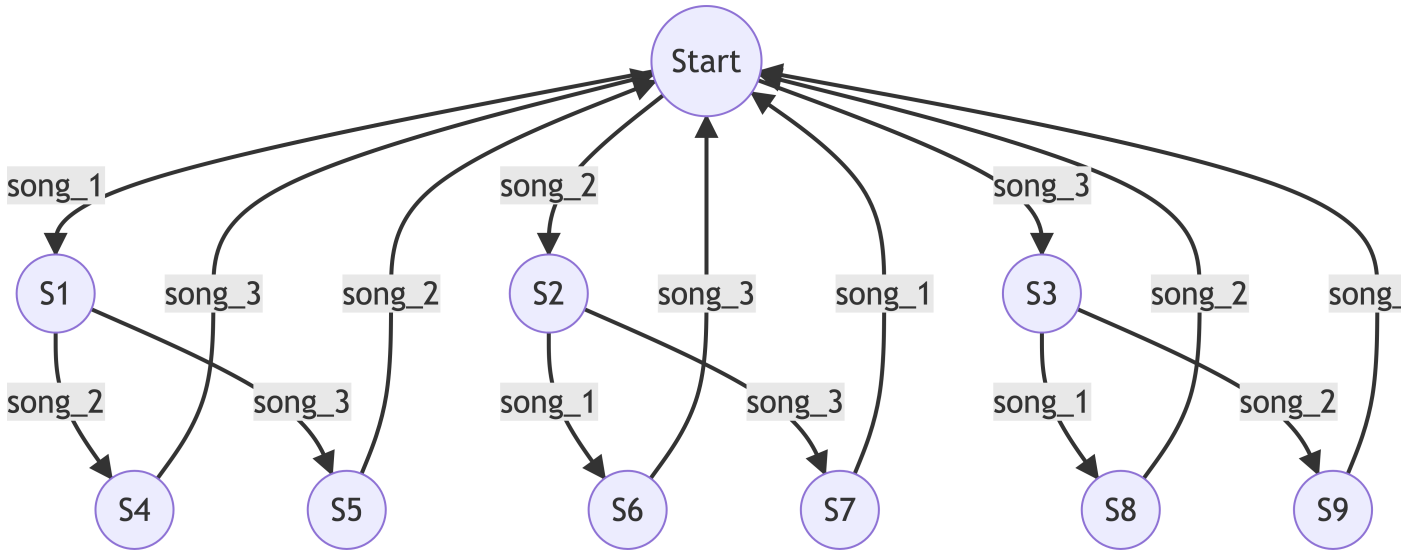
7. **ATM Transactions**: Design a DFA that accepts sequences where a user inserts a card, enters a PIN, selects an account, chooses a transaction type, and then takes the cash or completes the transaction.

[L = { w in {card, PIN, account, transaction, cash}* | w { starts with card and ends with cash or transaction} }]



8. **Music Playlist Shuffle**: Design a DFA that shuffles songs without repeating the same song until all songs in the playlist have been played.

[L = { w in {song_1, song_2, ... song_n}* | w { does not repeat any song until all songs have been played} }]


Start
song_1
song_2
song_3
S1
S2
S3
song_3
song_2
song_3
song_1
song_2
song_
song_2
song_3
song_1
song_3
song_1
song_2
S4
S5
S6
S7
S8
S9