finite state machine.md

Notes from video: [computerphile]

Finite state machines don't have memory. They remember the state they are at but don't recall how they got there. In the video, there's an example of a parking meter who needs to accept $0.25 before outputting a receipt for the driver to put on their dashboard. Any combination of dimes, nickels, or quarters are accepted, as long as it adds up to $0.25 but the meter only knows it how many cents it has at a given point in time.

Example elaborated: I give the meter $0.05, it knows its state is $0.05. I give the meter $0.10, it knows its state is $0.15. It **does not know what coins it was given to reach this state** simply that it's there. The combinations of coins to get the target number ($0.25) don't matter as long as the meter gets the correct number of coins.

```
new_state = current_state + input_value
# example of parking meter
new_state = 0.05 + 0.20
print(new_state) # 0.25
```

Notes from lecture [FOCS class]:

Definition: A finite state machine (or finite automaton) is a tuple M = (Q, ∑, Δ, s, F):

- Q is the finite set of states
- ∑ is the finite alphabet of a finite state machine
- Δ is the translation relation that links one state to another.
    - notation for translation relation: (p, a, q) ϵ Δ. In other words, p is state, a is link to state, q is next state and they're elements of the translation relation in a FSM.
- s ϵ Q is the start state. Start state is an element of the finite set of states.
    - **element of: refers to a single item within a set. start state is an element of (single item) of the finite set of states.**

- F is $\subseteq$ Q set of final states.
    - **subset of: refers to a collection of items in a set. there can be multiple final states in a finite set of states.**

Deterministic FSA: A FSA is deterministic if for every state q and every symbol Δ there is at least one state p. *The translation relation for a DFSA is a partial function because it's a function not defined on all parts of its domain.*

Additionally, if a DFA accepts a language A then the language is regular. Because, FSA only accept regular languages. But in the inverse case where we know a language A is regular, we don't know what FSA it will be entered into, ie - will it be a deterministic finite state machine or a non-deterministic one?

Hence the theorem: A language A is accepted by some finite automaton M if and only if A is regular.

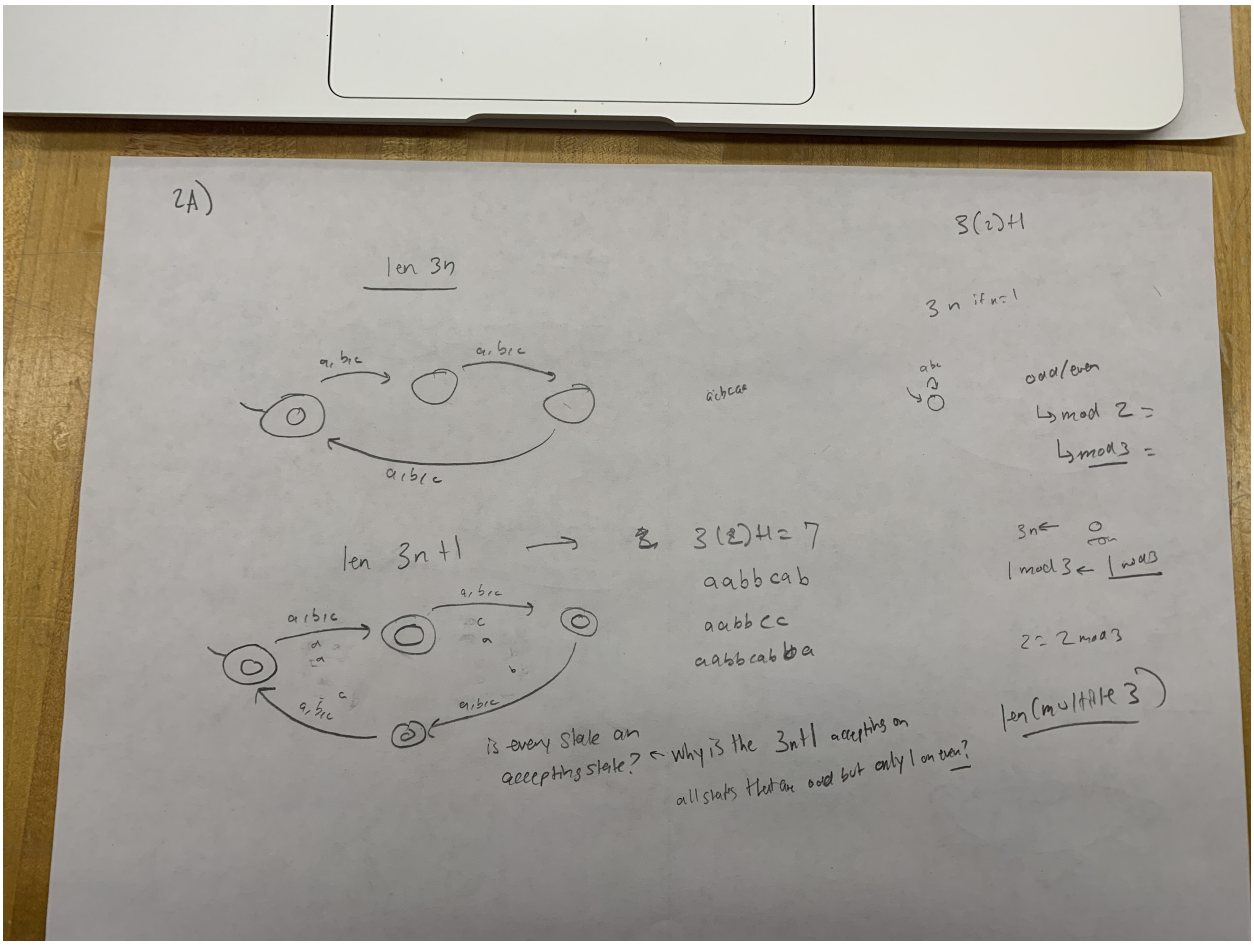
Non-deterministic FSA: A FSA is non-deterministic if it can move to any combination of states around it.

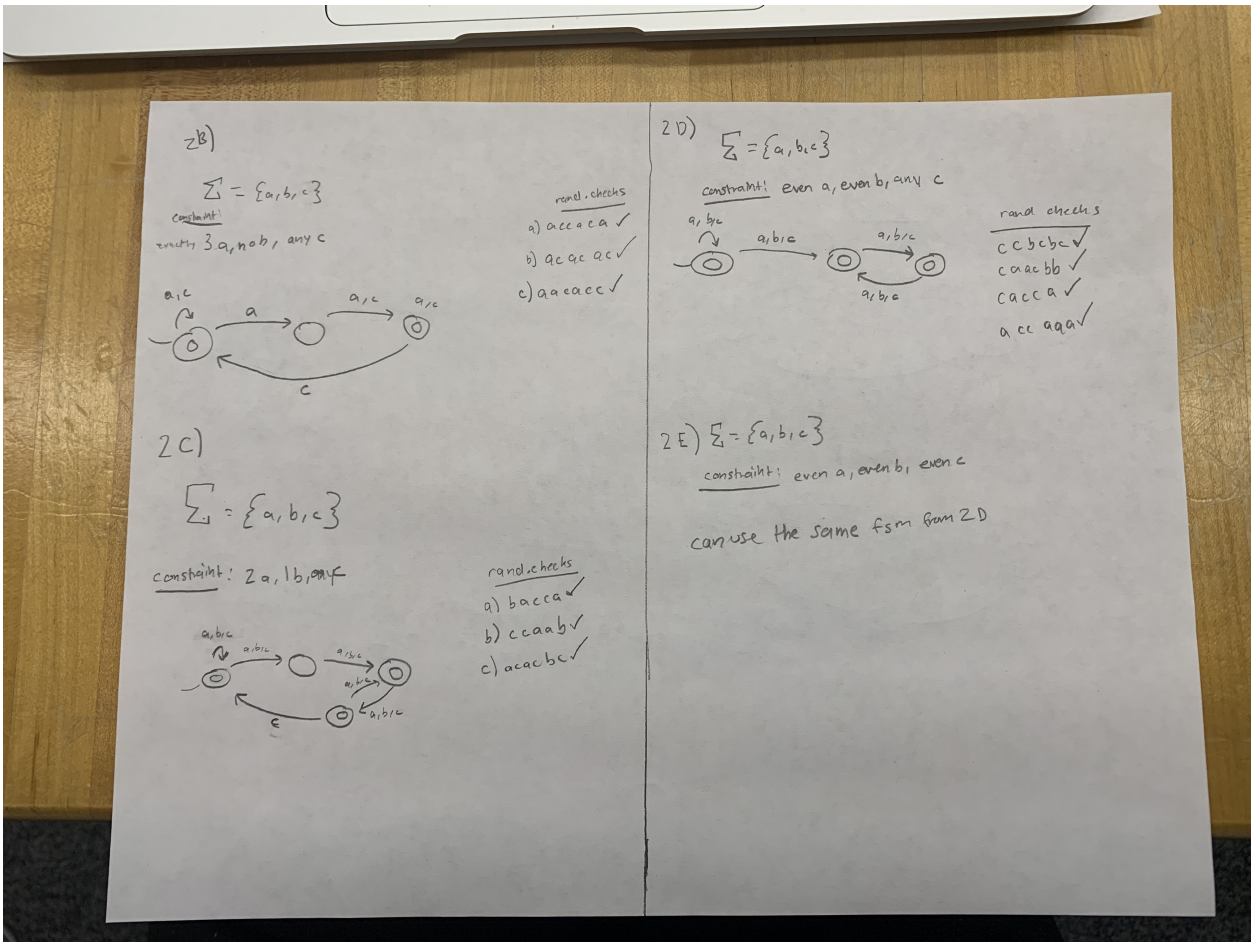
Given a non-deterministic FSM, we can create a deterministic FSM with the same language.

example finite state machine

The start state can be either circle and there is no final state, therefore the only input the FSM can accept is the $\varnothing$ (empty set).

Homework 2A - 2E FSM:


2A)
len 3n
a, b, c
a, b, c
a, b, c
len 3n+1
a, b, c
a, b, c
a, b, c
a, b, c
3(2)+1
3n if n=1
odd/even
mod 2 =
mod 3 =
3(2)+1=7
aabbcab
aabbcc
aabbcabba
3n
1 mod 3
1 mod 3
2 = 2 mod 3
len (multiple 3)
is every state an accepting state? ← why is the 3n+1 accepting on all states that are odd but only 1 on even?


2B)
Σ = {a, b, c}
constraint: exactly 3 a, no b, any c
a, c
a
a, c
a, c
c
rand. checks
a) acacaca ✓
b) acacacc ✓
c) aacaacc ✓
2C)
Σ = {a, b, c}
constraint: 2a, 1b, any c
a, b, c
a, b, c
a, b, c
a, b, c
ε
a, b, c
rand. checks
a) baccaa ✓
b) ccaaab ✓
c) acacbc ✓
2D)
Σ = {a, b, c}
constraint: even a, even b, any c
a, b, c
a, b, c
a, b, c
a, b, c
rand. checks
ccbcbc ✓
caacbb ✓
caccca ✓
acc aqqa ✓
2E) Σ = {a, b, c}
constraint: even a, even b, even c
can use the same fsm from 2D