Jan 13
2026

Non-deterministic Finite Automata

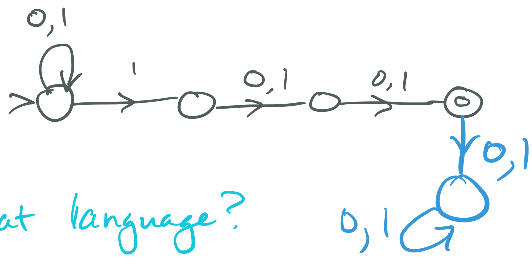
So far, we have required that for every $\sigma \in \Sigma$, and every state in our FA, we have a transition that will tell us what to do when we see σ in that state.

What if we allow 0 or more options of what to do on seeing σ while in state q?

Such an automaton is called a Non-deterministic Finite Automaton (NFA).

- a DFA is a special case of an NFA.

NFAs - Non-deterministic FAs.

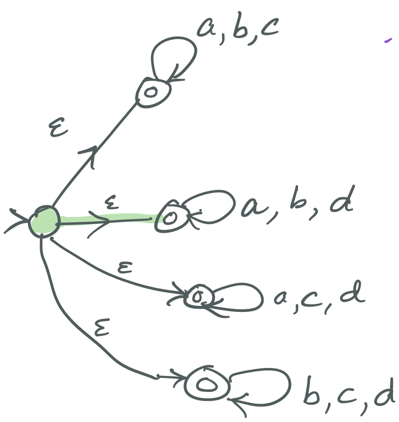


"third-to-last letter is 1"

What language?

The NFA "lucky-guesses" when it is about to see the third-last symbol, and verifies that the letter is indeed a 1.

Another thing a NFA can do is make a transition on ε.



What language?

$\Sigma = \{a,b,c,d\}$

$\{ w \in \{a,b,c,d\}^* \mid$

$\}$

Construct a NFA for "either ends in ab or $\#_a(w)$ is odd"

Def$^{\underline{n}}$ A NFA is a 5-tuple $(Q, \Sigma, \delta, s, F)$

where:

Q is a finite set of states

Σ is an alphabet

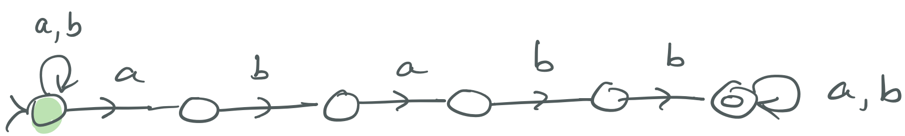
$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ is transition function

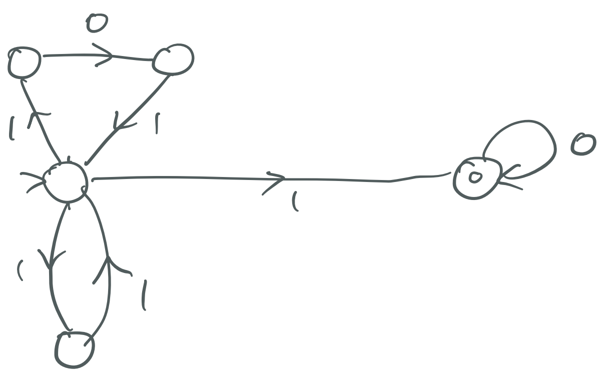
$s \in Q$ is start state

$F \subseteq Q$ is set of accept states.

"exists"

Recall - an NFA accepts a string if $\exists$ a computation on that string that ends at a final state ... there might be other computations that end at non-final states

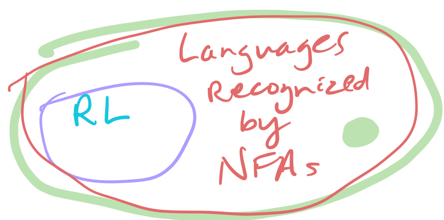




what language?

for FAs: Does non-determinism make our model more powerful? I.e. are there languages we can program a NFA to recognize that no DFA can recognize?

[Note: DFA is a special case of NFA ...]



An NFA that has no ε-transitions and has 0 or 1 choice of what to do in each state on each symbol is a DFA.

Theorem 1.39 ∀ NFA has an $\boxed{\text{equivalent}}$ DFA

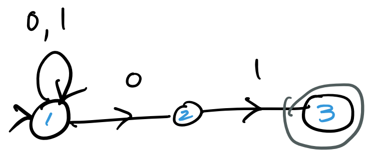
Note: by "equivalent", we mean "recognizes the same language".

Proof: Once again, I am going to give you a construction that converts a NFA into a DFA ... You should convince yourself it works

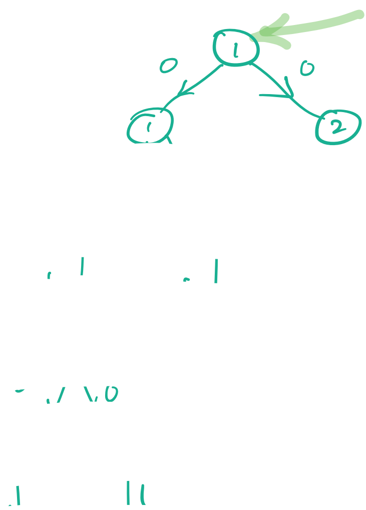
Or read the more detailed proof in text.

Idea: A DFA, on input w, has a computation sequence $(q_0, \sigma_1 \cdots \sigma_n) \vdash (q_1, \sigma_2 \cdots \sigma_n) \vdash \cdots \vdash (q_n, \varepsilon)$

What does a NFA have, on input w?
$\exists$ many possible paths the computation can take.
We can represent them in a tree, something like a decision tree



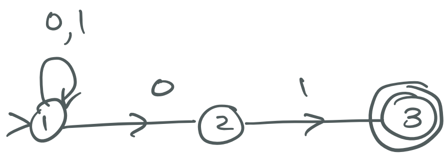
input = 00101

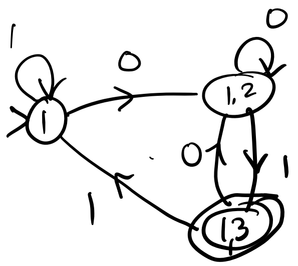


We are going to design a DFA that simulates all possible computations at once...
We will use our DFA-state to keep track of all possible states we could be in, in the NFA, at that

particular point in consuming the input
- after reading ε, we can be in state 1
- after reading 0 we can be in 1 or 2
- after reading 00 we can be in 1 or 2
⋮
etc

"ends in 01"

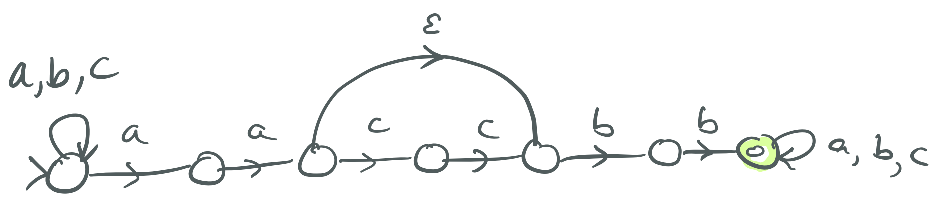




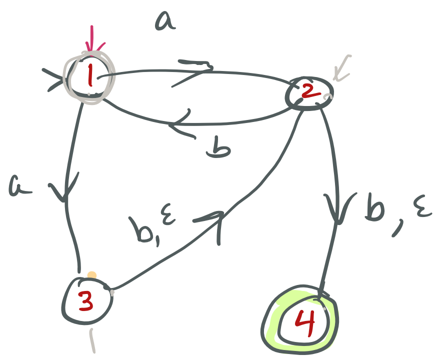
	0	1
1		
2		
3		

A look at how to use ε-transitions

Eg "contains aabb or aaccbb" $\Sigma = \{a, b, c\}$



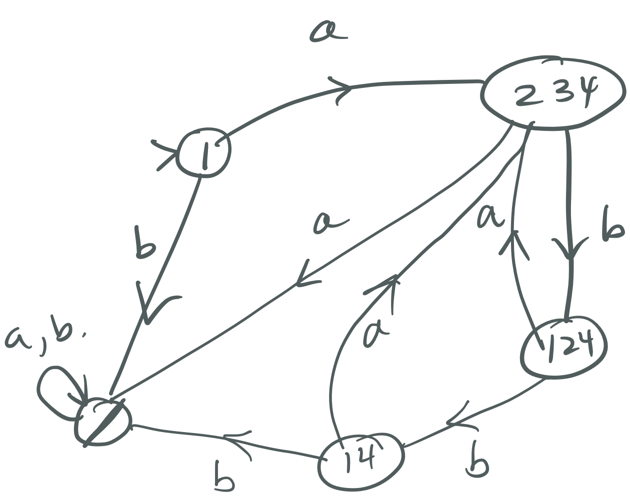
A smaller example for conversion to DFA.



	a	b
> 1		
2		
3		
4		

Rule:
In building the DFA table consider all possible states you could get to on reading an "a", say, and then with 0 or more ε-transitions.

Remember to mark start state and Final states

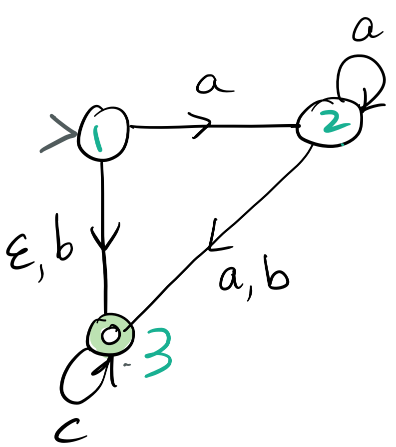


The above-given method converts any NFA into an equivalent DNA.

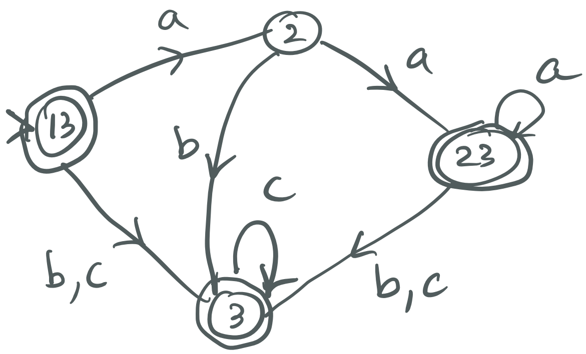
obvious when you think about it...

because the computation path in the DFA will carry all possible computation paths in the original NFA.

How do we handle ε-transitions from start state?



	a	b	c
1	2	3	∅
2	23	3	∅
3			



Start state contains all states you can get to from original state using only ε-transitions.

Now that we know that every NFA also recognizes a language that is Regular

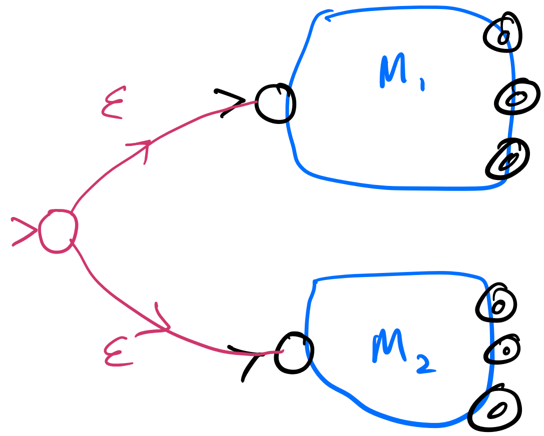
is recognized by some DFA

We can prove more easily:...

Theorem 1.45 RL is closed under $\cup$

Proof: Let L_1 and L_2 be RLs.

$\therefore$ $\exists$ M_1 and M_2, DFAs that recognize L_1 and L_2, respectively.



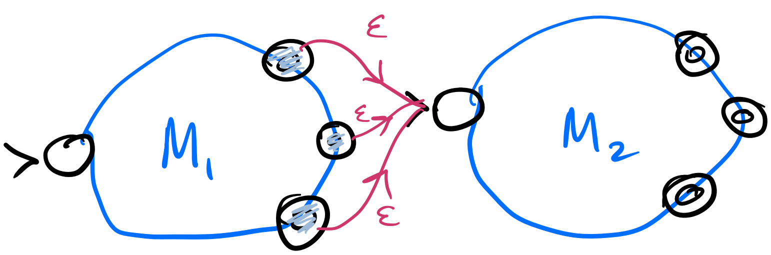
add new start state and the red edges.

∎

Theorem 1.47 RL is closed under $\circ$

Proof: Let L_1 be a RL, recognized by FA M_1
L_2 " " " " FA M_2

Then we construct the following NFA M



M

add red edges and make M_1's final states NOT final

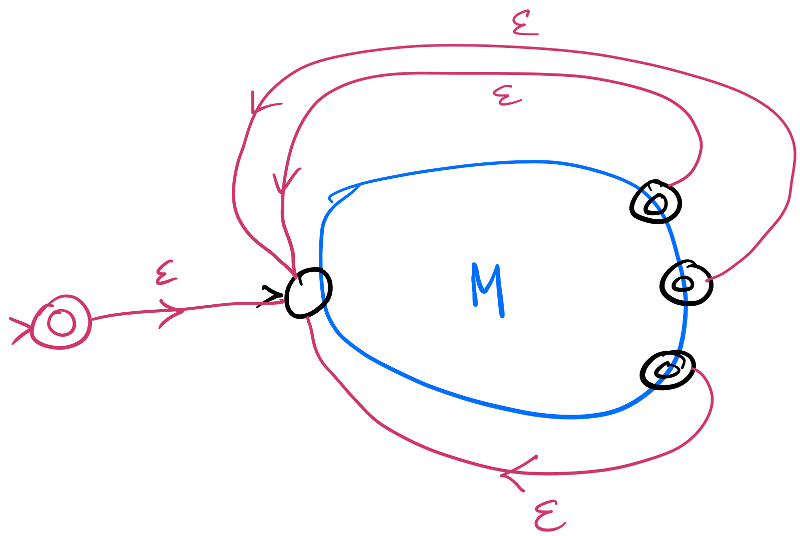
M recognizes $L_1 \circ L_2$

Theorem 1.49 RL is closed under *

Proof: Let L be any language in RL.

By def$^{\underline{n}}$ of RL, $\exists$ a FA M that recognizes L.

We construct a new NFA M′ from M as follows:



- add the red edges and a new start state

M′ recognizes L^*. ∎

Q. Why did we add a new start state? What kind of trouble could we get into if we just made the old start state into a final state? Come up with a FA that would lead us into that trouble.