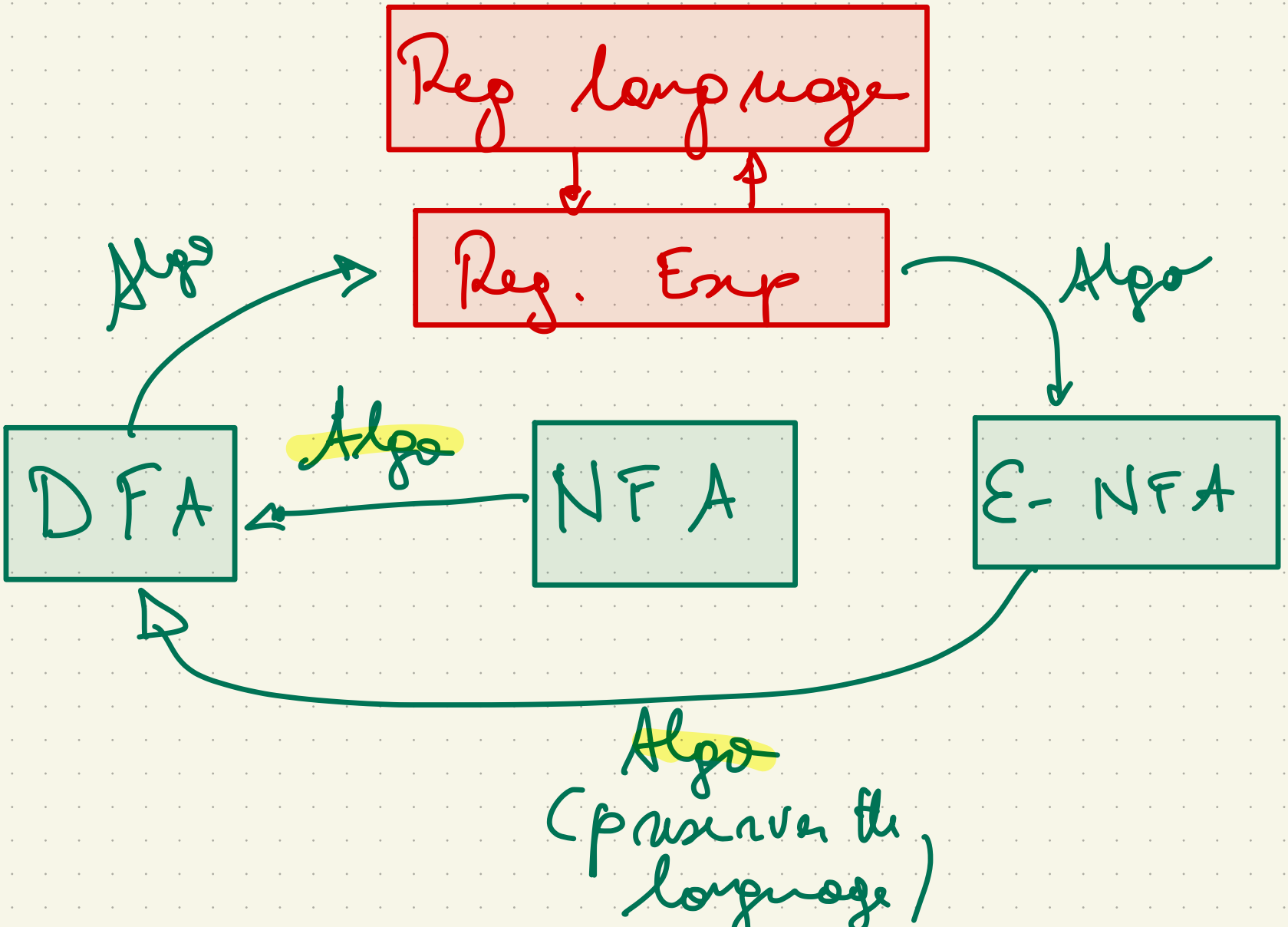
September, 26th

Kleene's theorem



Reg. Exp → ε-NFA

1. The constant $\emptyset$. It denotes the language $L(\emptyset) = \emptyset$.
2. The constant ε. It denotes the language $L(\varepsilon) = \{\varepsilon\}$.
3. All constants $a \in \Sigma$. Each constant $a \in \Sigma$ denotes the language $L(a) = \{a\}$.

1. 

$L(A) = \emptyset$

2. 

$L(A) = \{\varepsilon\}$

3. 

$L(A) = \{a\}$

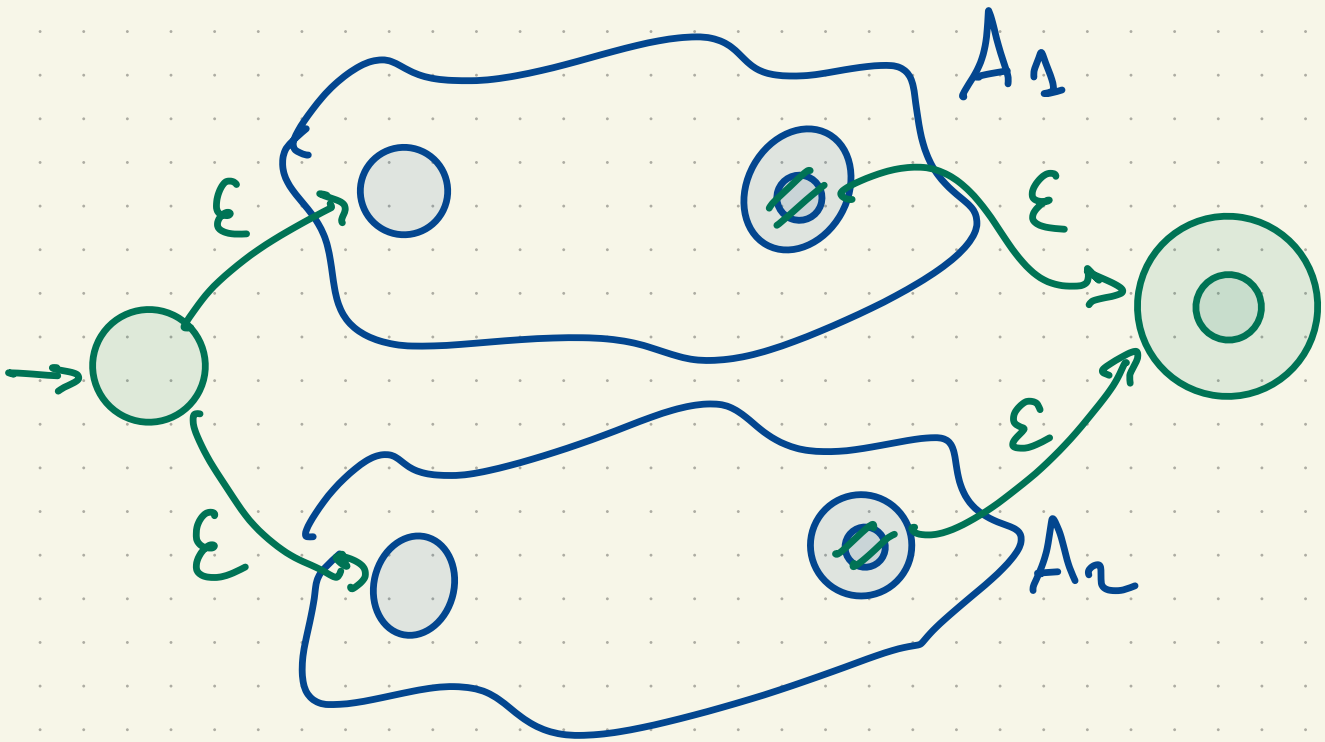
4. All expressions of the form $r_1 + r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 + r_2$ denotes the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.

$r_1 + r_2$

$L(A_1) = L(r_1)$

$L(A_2) = L(r_2)$

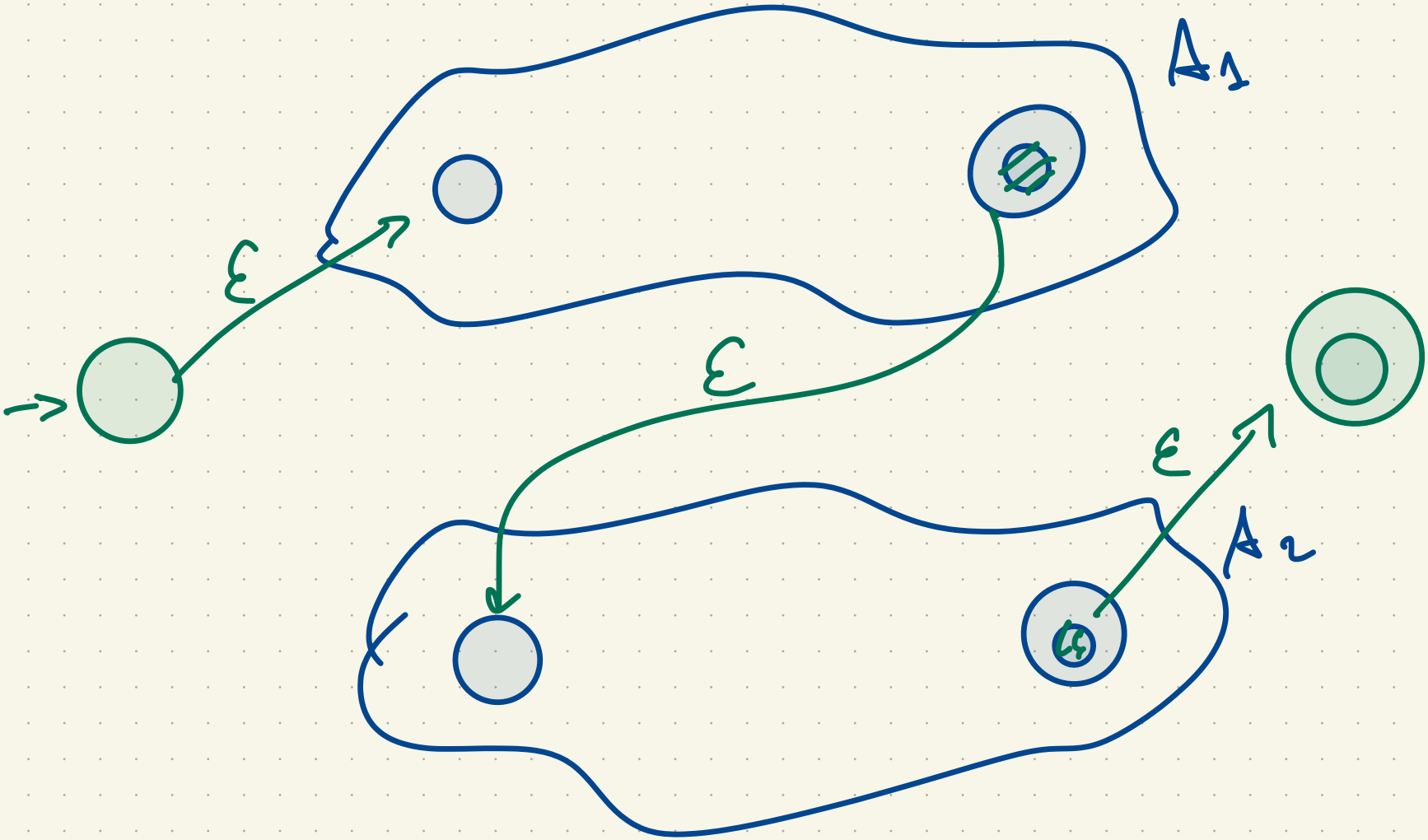
4.



5. All expressions of the form $r_1 \cdot r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 \cdot r_2$ denotes the language $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.

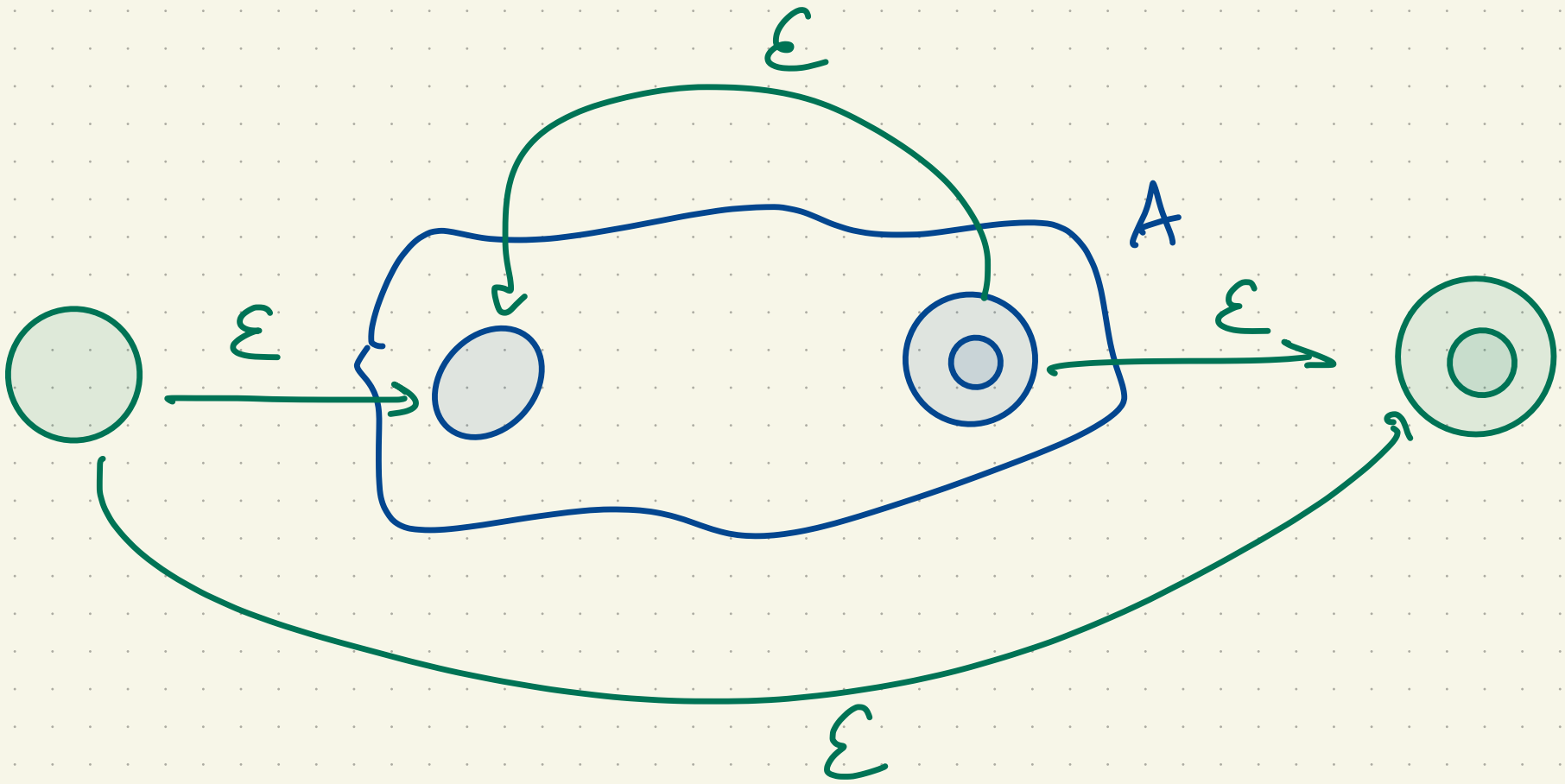
$L(A_1) = L(r_1)$

$L(A_2) = L(r_2)$



6. All expressions of the form r^*, where r is a regular expression on Σ.
   Each expression r^* denotes the language $L(r^*) = \big(L(r)\big)^*$.

$L(A) = L(r)$

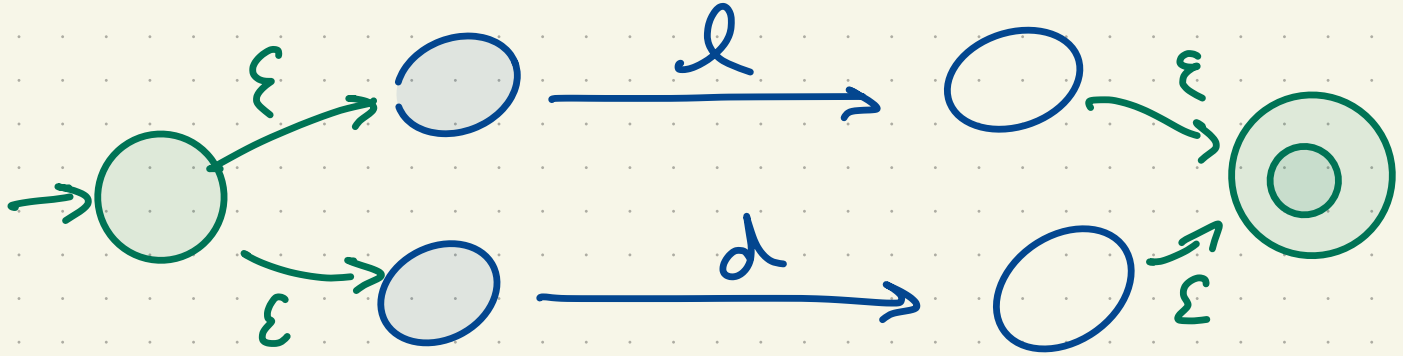


Example: $l.(l+d)^*$

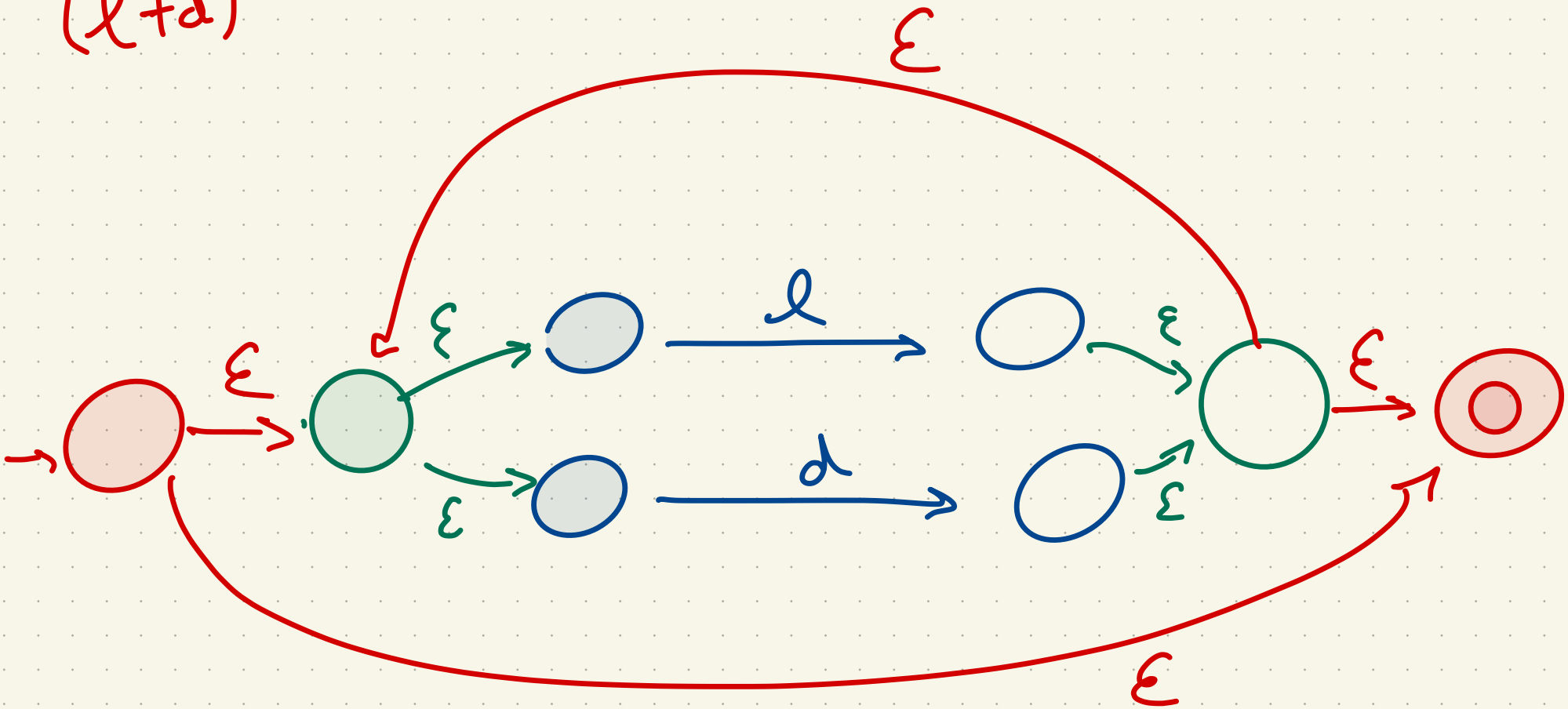
l

d

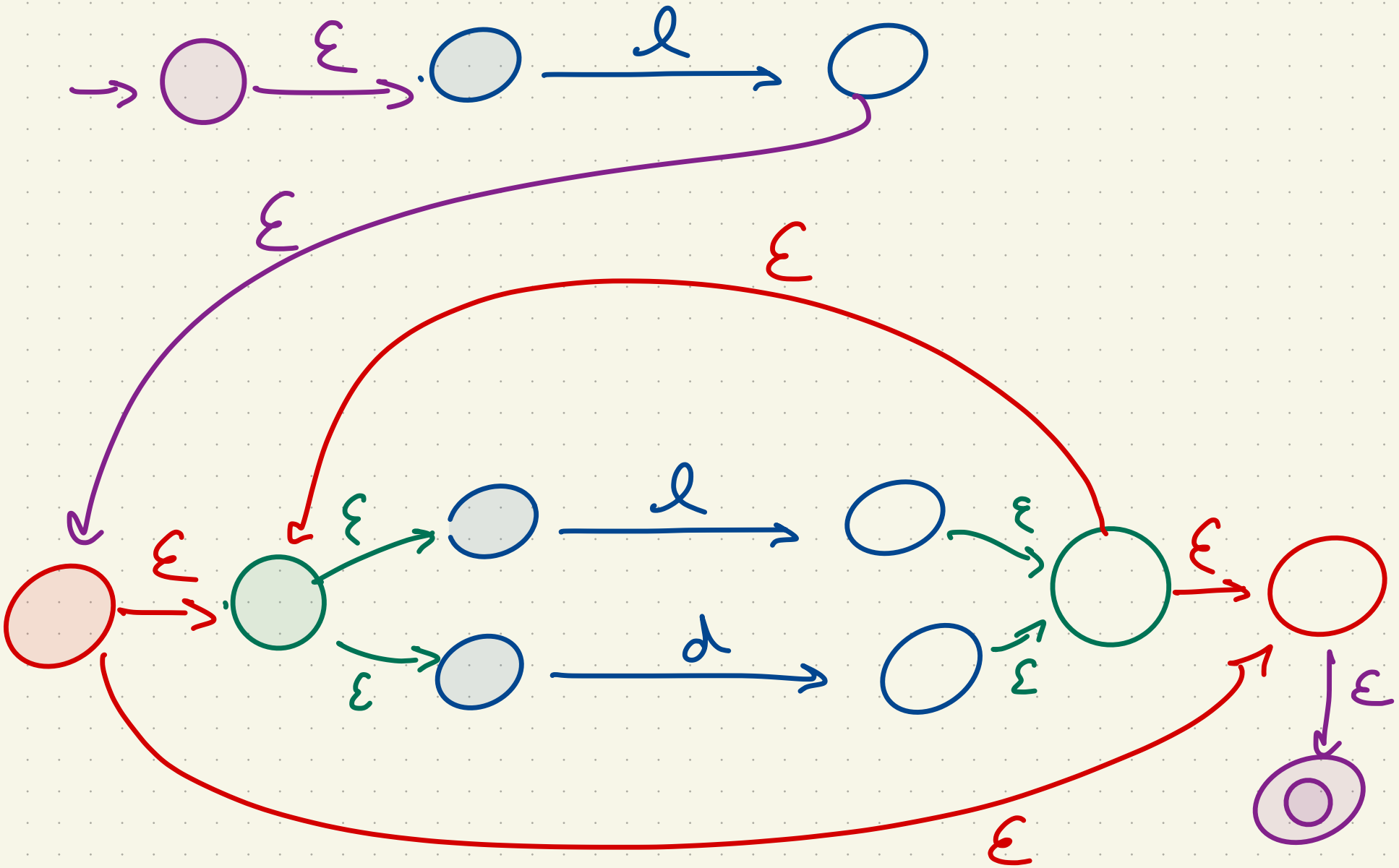
$l+d$



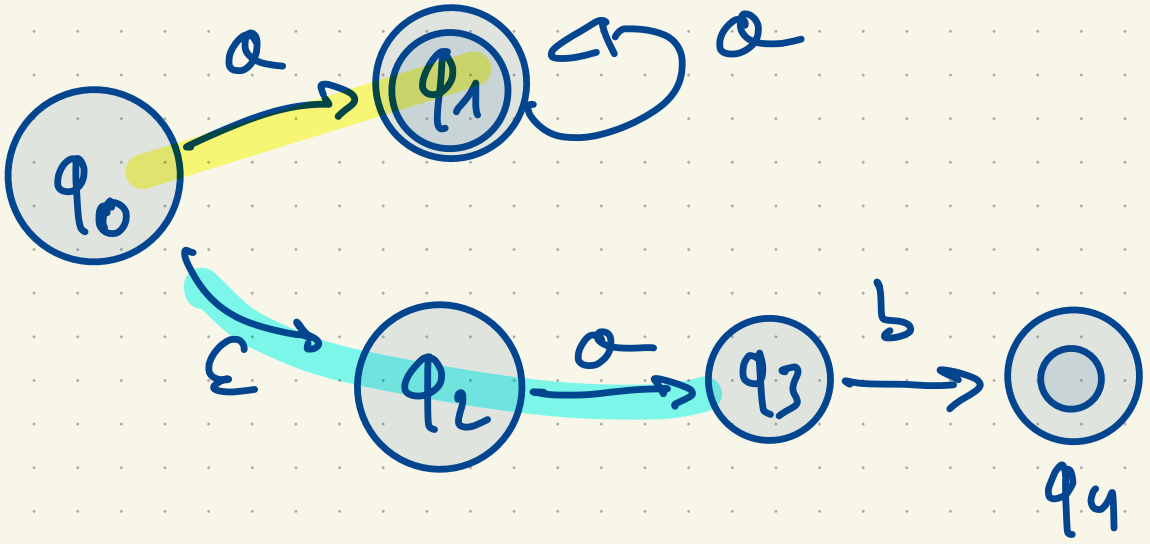
$(l+d)^*$



$l . (l + d)^*$



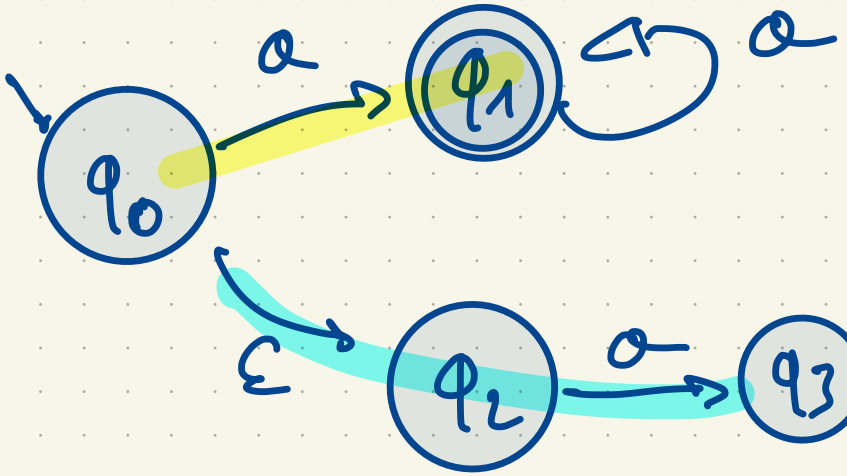
From ε-NFAs to DFAs.



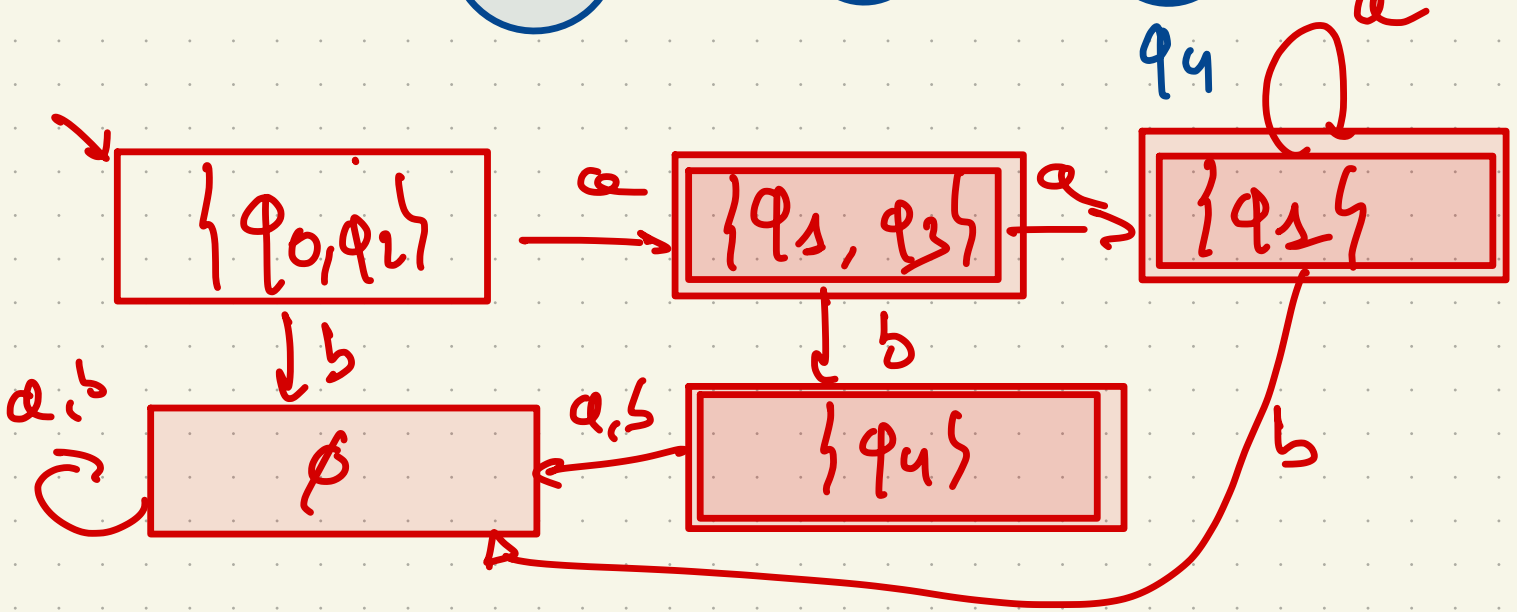
The automaton can have several runs on the same word

$w = a$ $\quad q_0 \to q_1$ $\quad q_0 \to q_2 \to q_3$

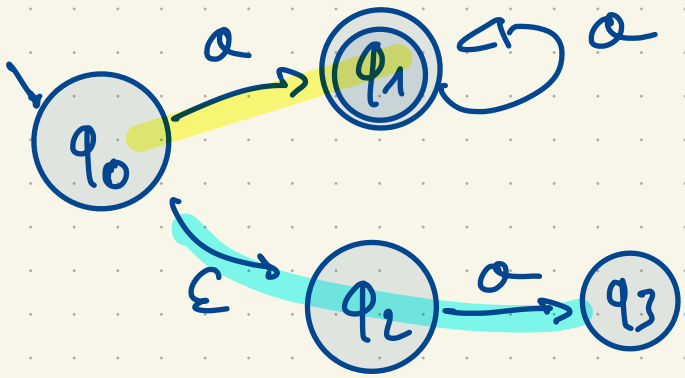
Main Idea: Build a DFA that simulates all the possible runs of the ε-NFA "in parallel".



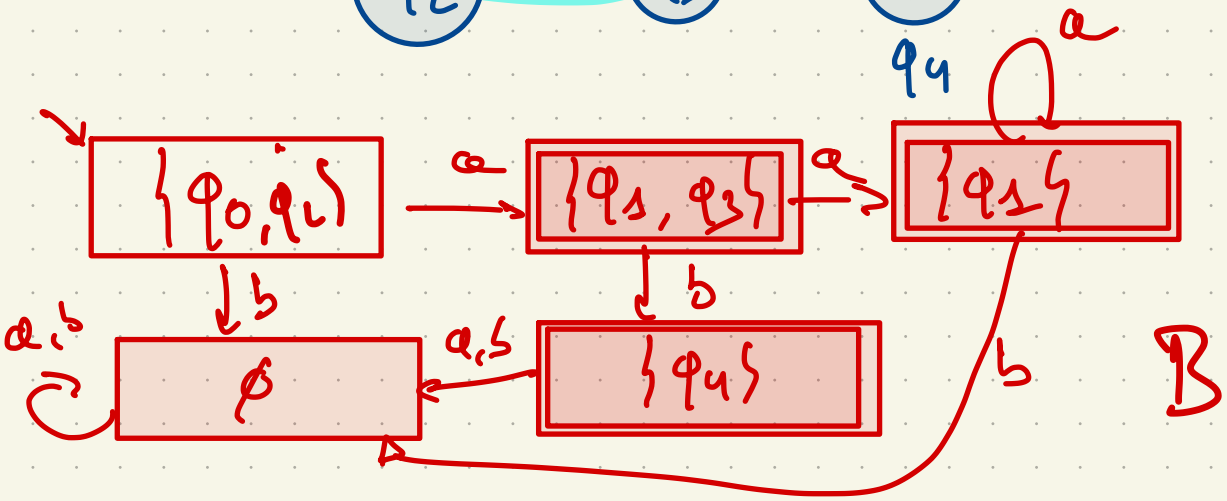
$\Sigma = \{a, b\}$



States of the DFA are subsets of the states of the ε-NFA.



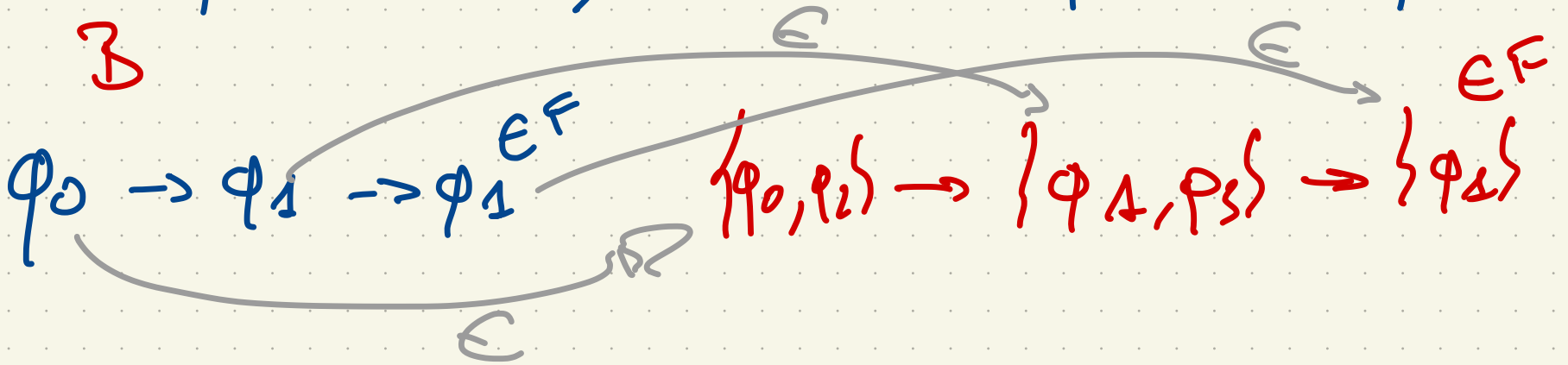
Why is this construction correct?

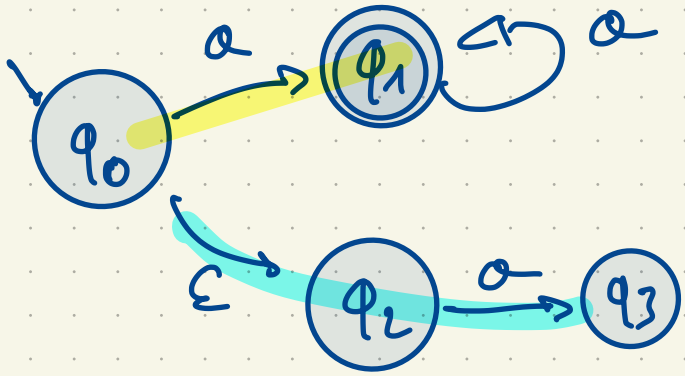


$L(A) \subseteq L(B)$

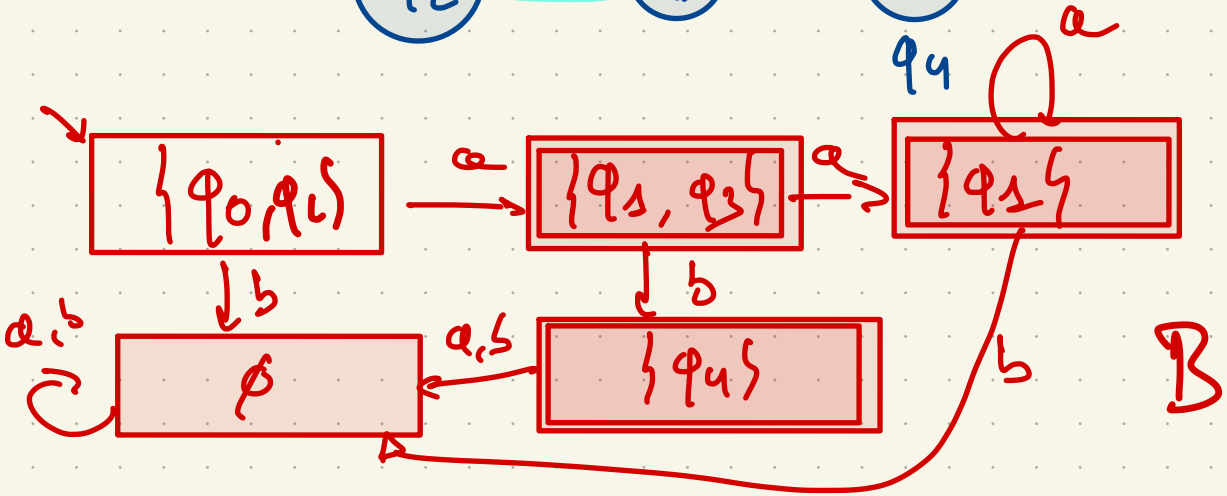
$\Uparrow$

For each path in A, there is an "equivalent" path in B





A



B

For each path in B, we can find at least one corresponding path in A

$\{q_0, q_2\} \to \{q_1, q_3\} \to \{q_4\}$

$q_0 \xrightarrow{\varepsilon} q_2 \to q_3 \longrightarrow q_4$

$\Downarrow$

$L(B) \subseteq L(A)$

Determinisation of ε-NFAs

Given an ε-NFA $A = \langle Q^A, \Sigma, \delta^A, q_0^A, F^A \rangle$, we build the DFA $D = \langle Q^D, \Sigma, \delta^D, q_0^D, F^D \rangle$ as follows:

1. $Q^D = 2^{Q^A}$
2. $q_0^D = \varepsilon\text{closure}(q_0^A)$
3. $F^D = \{S \in Q^D \mid S \cap F^A \neq \emptyset\}$
4. for all $S \in Q^D$, for all $a \in \Sigma$: $\delta^D(S, a) = \varepsilon\text{closure}(\delta^A(S, a))$

2^{Q^A} = set of all subsets of Q^A

ε-closure(q) = {all states reachable from q by reading ε}

Remark: If I have n states in my NFA

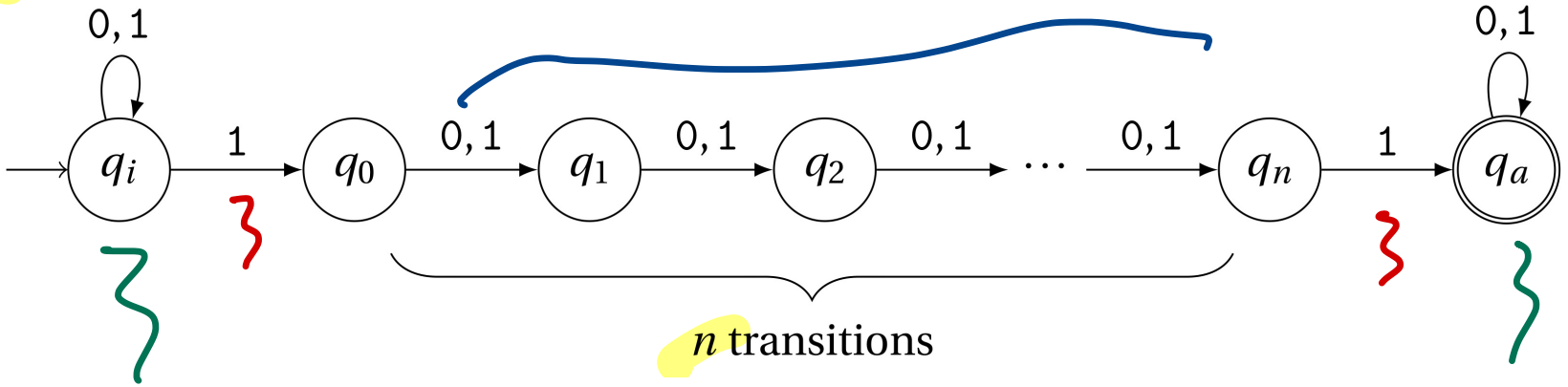
I can have up to 2^n states in my DFA

example: $Q = \{q_1, q_2, q_3\}$

$$2^Q = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\}$$

Unfortunately this is unavoidable!

An



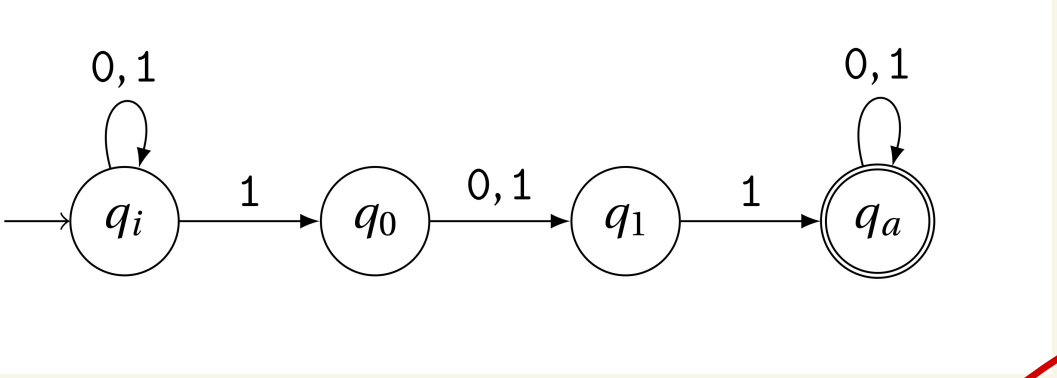
0 1 1 0 1 | 1 | 0 1 1 0 1 0 1 | 1 | 0 1 1 0 0

prefix — n bits — suffix

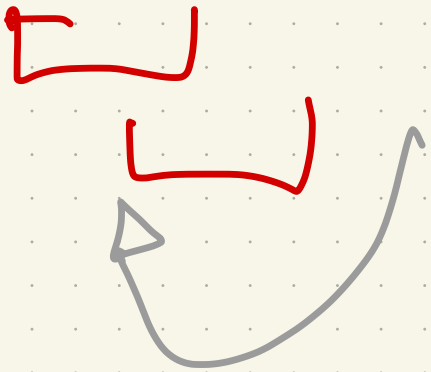
n	size of A
1	4
2	5
3	6

→ linear growth.

Let's fix $n = 1$



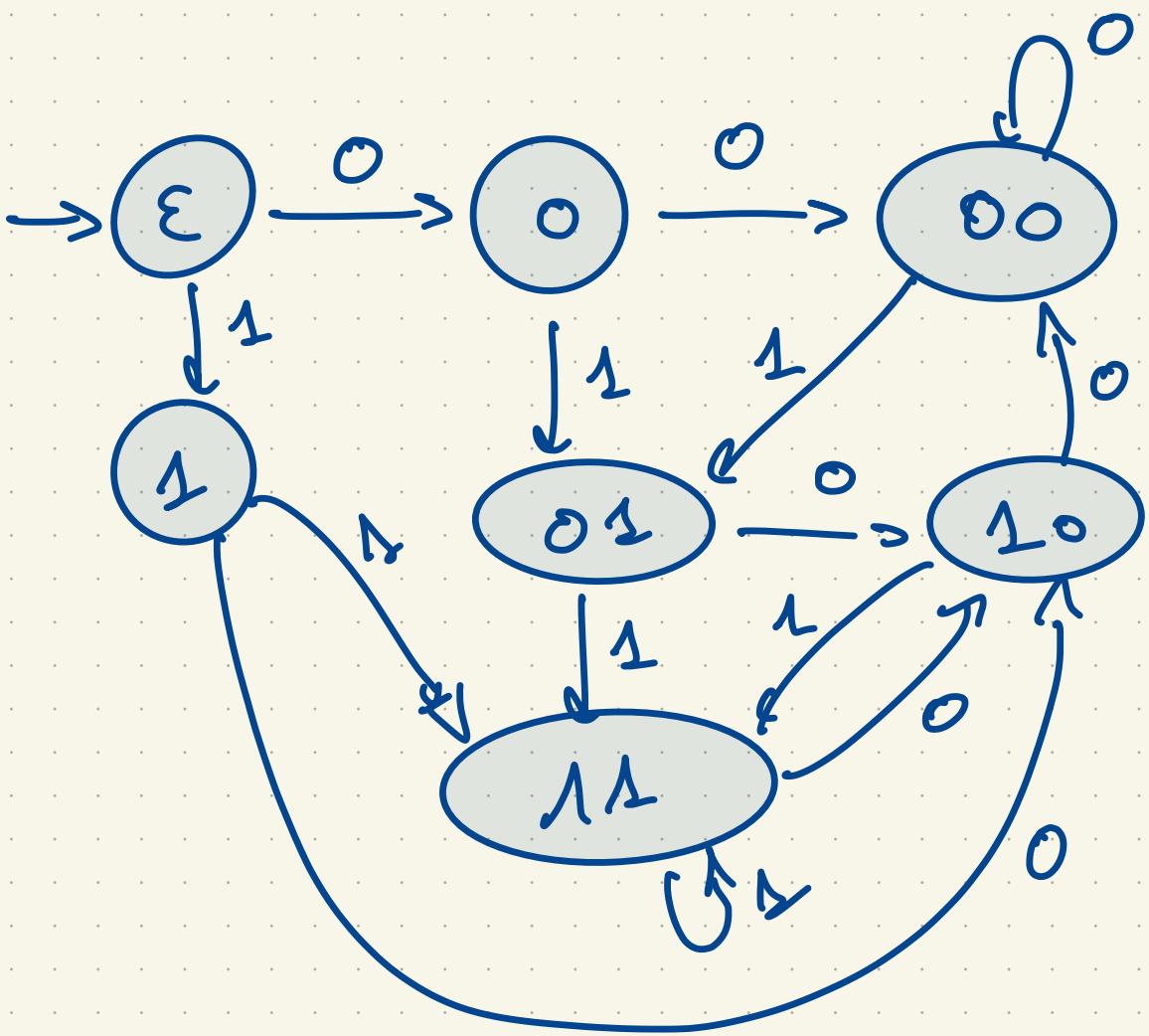
1 0 0 1 0 1



Idea: keep "in memory" the two last bits

How can I "remember" the two last bits?

You store the memory in states



→ at least $2^2 = 4$ states

n	≥ states
1	≥ 4
2	≥ 8
⋮	
10	≥ 1024

In the DFA, I need
at least $2^{(n+1)}$ states
to store the memory
that I need to recognize
the language.