Abbiamo visto, in ordine, le espressioni regolari; poi, come con la costruzione di Thompson, si possa ottenere un NFA che riconosca esattamente il linguaggio denotato da quell'espressione regolare.
Oggi vedremo la simulazione di un NFA, cioè useremo l'NFA per verificare se accetta una certa parola che gli diamo in input.

Un NFA N **accetta** la parola w, se e solo se esiste un cammino $x_1 \dots x_k = w$ che parte dallo stato iniziale e termina in uno stato finale. La definizione mette in luce diversi gradi di non-determinismo: per una stessa parola, vi possono potenzialmente essere più di un cammino etichettato. Se eventualmente vogliamo identificare tutti i cammini, possiamo adottare due strategie. Una consiste nell'utilizzare una tecnica di backtracking. Esiste tuttavia una strategia più efficiente, che ha a che fare con la ε-closure.

Supponiamo di avere l'NFA $N = (S, A, move_n, s_0, F)$ con $t \in S$
La ε-closure($\{t\}$) è l'insieme degli stati in S che possono essere raggiunti da t tramite 0 o più ε-transizioni (transizione etichettata da ε).
↑
Fondamentale: la ε-closure di $\{t\}$ può essere vuota? NO, $t \in \varepsilon$-closure($\{t\}$)

$T \subseteq S$ $\quad \varepsilon\text{-closure}(T) = \bigcup_{t \in T} \varepsilon\text{-closure}(\{t\})$ ossia: - tutti gli stati in T
- e tutti gli stati che possono essere raggiunti da essi con ε-transizioni.

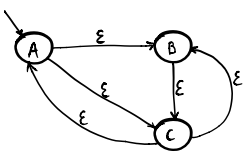
COMPUTAZIONE DI ε-closure($\{t\}$)

$N = (S, A, move_n, s_0, F)$
$t \in S$
STRUTTURE DATI DI CUI CI AVVALIAMO:
- Pila: per le computazioni parziali
- Array booleano "alreadyOn" per sapere in tempo costante se gli stati sono nella pila
- Array bidimensionale per memorizzare la funzione $move_n$, di dimensione $|S| \times (|A|+1)$ in cui ogni entry (t,x) è una lista linkata che rappresenta $move_n(t,x)$.
↑ colonna di ε

ALGORITMO

```
alreadyOn = {FALSE ... FALSE};   % INIZIALIZZAZIONE
closure (t, stack) {
    stack.push(t);
    alreadyOn[t] = TRUE;
    for (stato u ∈ move_n(t,ε)) {
        if (!alreadyOn[u])
            closure(u, stack);
    }
}
```

ESECUZIONE:



$t \in A$

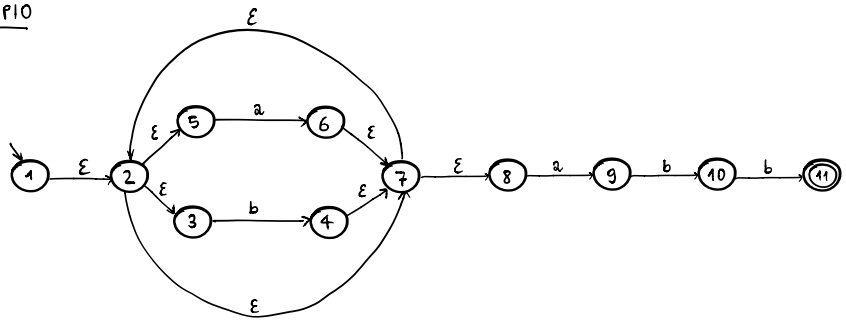
LA PILA RISULTANTE È ABC o ACB

SIMULAZIONE DI NFA

INPUT: $w\$$ con $\$ \notin A$, NFA $N = (S, A, move_n, s_0, F)$
OUTPUT: YES/NO se $w \in L(N)$ o $w \notin L(N)$

```
1.  states = ε-closure({s0})
2.  symbol = nextchar()   % input buffer
3.  while (symbol ≠ $) {
4.      states = ε-closure(⋃_{t ∈ states} move_n(t, symbol))   % Per ogni stato t ∈ states, guardo dove mi muovo col symbol che sto leggendo
5.      symbol = nextchar()
6.  }
7.  if (states ∩ F ≠ ∅) return YES; else return NO
```

ESEMPIO



w = abaabb ∈ linguaggio?

INPUT: w$ = abaabb$

ESECUZIONE:

1.	states = ε-closure ({ s_0 })	states = ε-closure ({1}) = {1,2,3,5,7,8} ← Sicuramente c'è perché lo sto chiudendo
2.	symbol = nextchar()	symbol = a
3.	while (symbol ≠ $) {	TRUE
4.	states = ε-closure ($\bigcup_{t \in states} move_n(t, symbol)$)	states = ε-closure ({6,9}) = {6,9,7,8,2,5,3}
5.	symbol = nextchar()	symbol = b
3.	while (symbol ≠ $) {	TRUE
4.	states = ε-closure ($\bigcup_{t \in states} move_n(t, symbol)$)	states = ε-closure ({10,4}) = {10,4,7,8,2,5,3}
⋮		
7.	if (states ∩ F ≠ ∅) return YES; else return NO	YES

Alla fine, otteniamo YES. YES indica che negli states ce ne è almeno uno finale, e quindi che c'è un cammino che porta allo stato finale.

Vi sono due approcci per l'utilizzo degli NFA:
- Utilizzarlo direttamente come riconoscitore di un linguaggio (pesante, perché deve calcolare ogni volta le chiusure);
- Trasformarlo in un DFA equivalente (che riconosca il medesimo linguaggio), che consente un'analisi più veloce (lineare).

Spesso non vale la pena di fare questa trasformazione. È il caso di grep, in cui non è prevista una forma di riciclaggio della chiusura che si costruisce.