Compilation Principle
编 译 原 理

第3讲：词法分析(3)

张献伟

xianweiz.github.io

DCS290, 3/1/2022

Quiz Questions

- Q1: lexical analysis of “if i == 0”?

  <keyword, ‘if’>, <id, ’i’>, <op, ‘==‘>, <num, ‘0’>

- Q2: usage of RE and FA in lexical analysis?

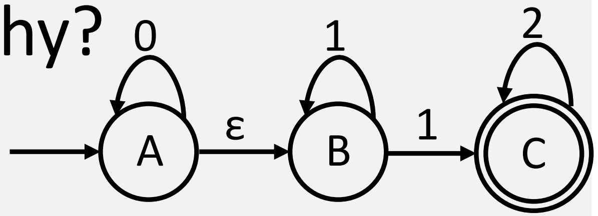
  RE: specify the token pattern; FA: implement the token recognizer

- Q3: write a regular expression for all strings of *a*s and *b*s which contains the substring *aba*

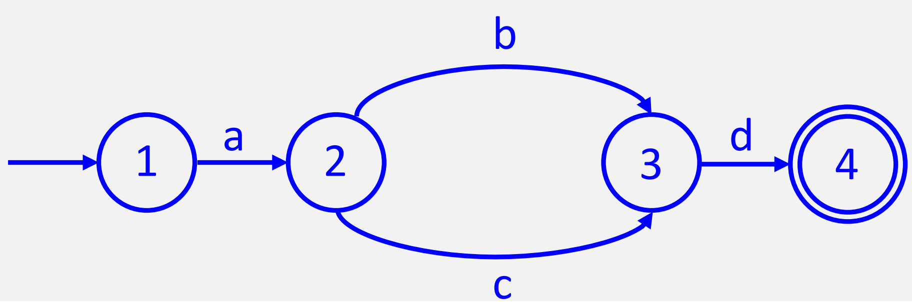
  (a|b)*aba(a|b)*

- Q4: the graph describes NFA or DFA? Why?

  NFA. A: ε-transition, B: 1-transition

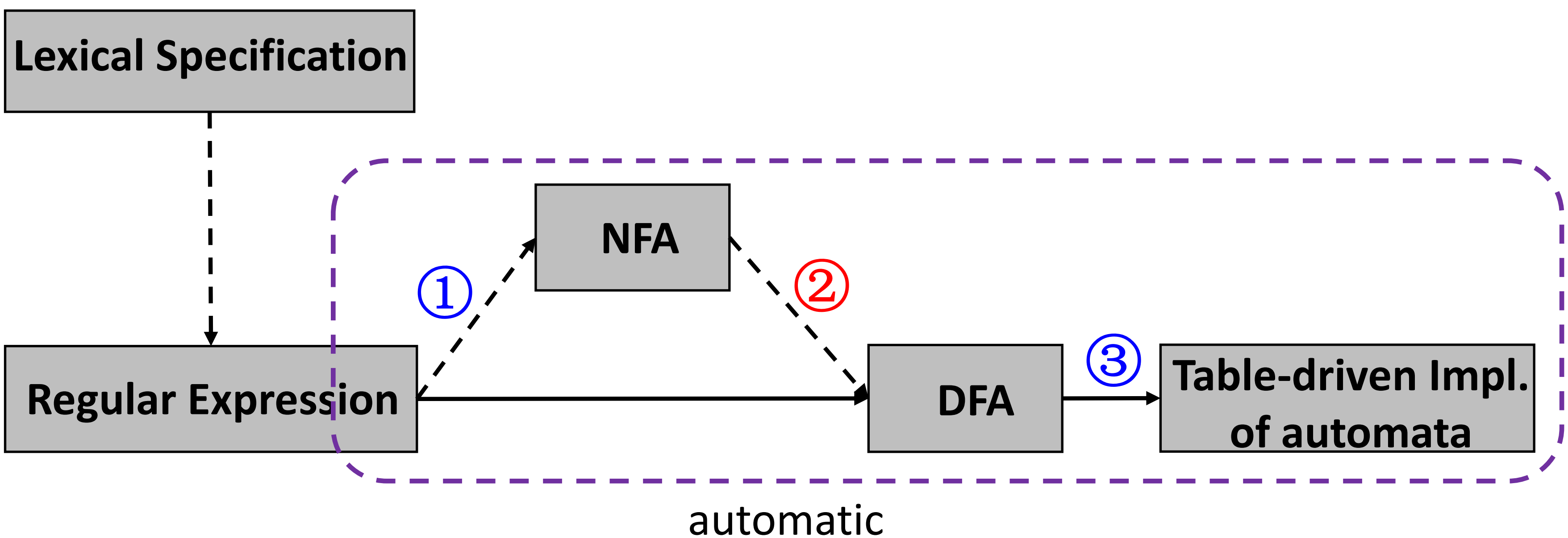


- Q5: FA for the Regex a(b|c)d



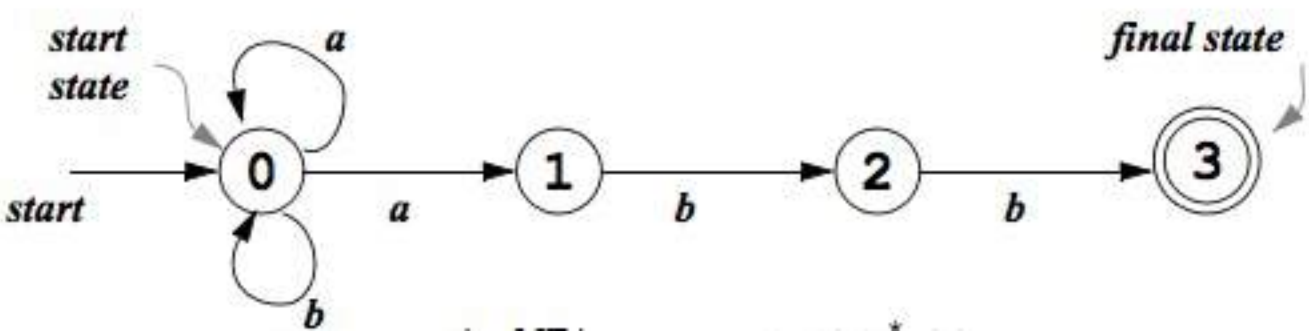
The Conversion Flow

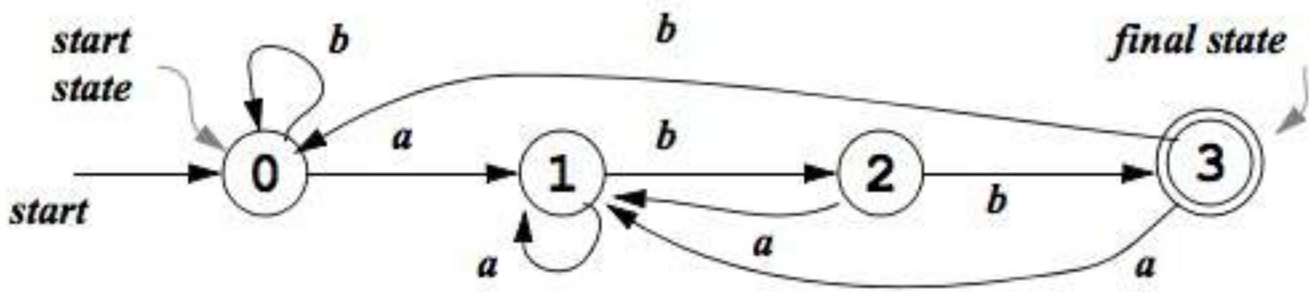
- Outline: RE → NFA → DFA → Table-driven Implementation
  - ③ Converting DFAs to table-driven implementations
  - ① Converting REs to NFAs
  - ② Converting NFAs to DFAs



NFA → DFA: Same[等价]

- NFA and DFA are equivalent





To show this we must prove every DFA can be converted into an NFA which accepts the same language, and vice-versa

NFA → DFA: Theory[相关理论]

- Question: is L(NFA) ⊆ L(DFA)?
  - Otherwise, conversion would be futile
- Theorem: L(NFA) ≡ L(DFA)
  - Both recognize regular languages L(RE)
  - Will show L(NFA) ⊆ L(DFA) by construction (NFA → DFA)
  - Since L(DFA) ⊆ L(NFA), L(NFA) ≡ L(DFA)

  Any DFA can be easily changed into NFA
- Resulting DFA consumes more memory than NFA
  - Potentially larger transition table as shown later
- But DFAs are faster to execute
  - For DFAs, number of transitions == length of input
  - For NFAs, number of potential transitions can be larger
- NFA → DFA conversion is done because the speed of DFA far outweighs its extra memory consumption

NFA → DFA: Idea

- Algorithm to convert[转换算法]
  - Input: an NFA *N*
  - Output: a DFA *D* accepting the same language as *N*

- **Subset construction**[子集构建]
  - Each state of the constructed DFA corresponds to a set of NFA states
    - Hence, the name ‘subset construction’
  - After reading input $a_1a_2\ldots a_n$, the DFA is in that state which corresponds to the set of states that the NFA can reach, from its start state, following paths labeled $a_1a_2\ldots a_n$

NFA → DFA: Steps

- The **initial state** of the DFA is the set of all states the NFA can be in without reading any input
- For any state $\{q_i, q_j, \ldots, q_k\}$ of the DFA and any input *a*, the **next state** of the DFA is the set of all states of the NFA that can result as next states if the NFA is in any of the states $q_i, q_j, \ldots, q_k$ when it reads *a*
  - This includes states that can be reached by reading *a* followed by any number of ε-transitions
  - Use this rule to keep adding new states and transitions until it is no longer possible to do so
- The **accepting states** of the DFA are those states that contain an accepting state of the NFA.

https://web.cse.msstate.edu/~hansen/classes/3813spring05/slides/04SubsetConstruction.pdf

NFA → DFA: Algorithm

```
Initially, ε-closure(s0) is the only state in Dstates and it is unmarked
while there is an unmarked state T in Dstates do
    mark T
    for each input symbol a ∈ Σ do
        U := ε-closure(move(T,a))
        if U is not in Dstates then
            add U as an unmarked state to Dstates
        end if
        Dtran[T,a] := U
    end do
end do
```

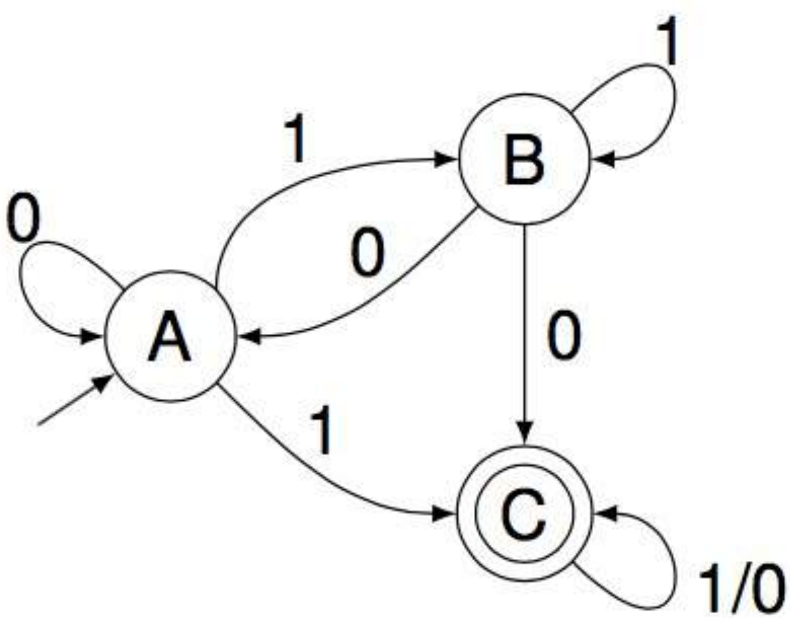
- Operations on NFA states:
  - ε-closure(s): set of NFA states reachable from NFA state ***s*** on ε-transitions **alone**
  - ε-closure(T): set of NFA states reachable from some NFA state ***s*** in set ***T*** on ε-transitions **alone**; $= \cup_{s \text{ in } T}$ε-closure(s)
  - move(T, a): set of NFA states to which there is a transition on input symbol ***a*** from some state ***s*** in ***T***

NFA → DFA: Example

- Start by constructing ε-closure of the start state
  - ε-closure(A) = A
- Keep getting ε-closure(*move(T, a)*)

T: A, *a*: 0/1

- Stop, when there are no more new states



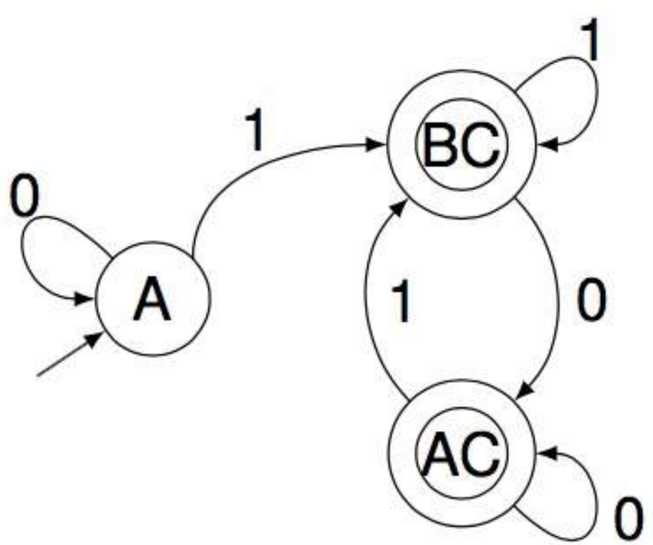
alphabet →

state ↓

	0	1
A	**A**	**BC**
BC	**AC**	**BC**
AC	**AC**	**BC**

NFA → DFA: Example (cont.)

- Mark the final states of the DFA
  - The accepting states of *D* are all those sets of *N*'s states that include at least one accepting state of *N*



alphabet →

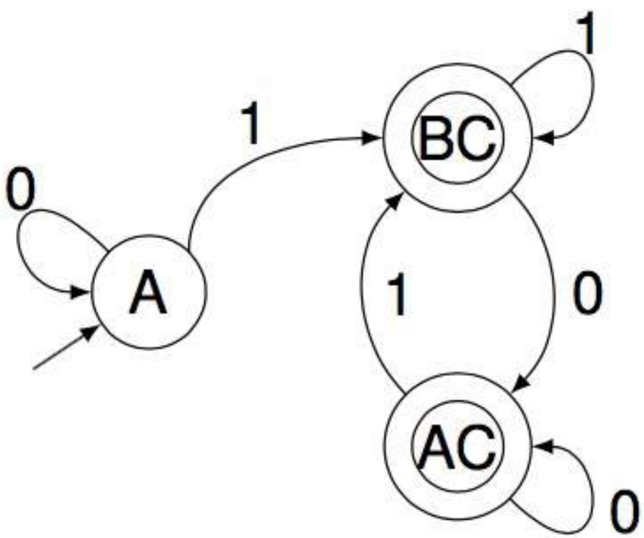
state ↓

	0	1
A	**A**	**BC**
BC	**AC**	**BC**
AC	**AC**	**BC**

- Is the DFA minimal?
  - As few states as possible

NFA → DFA: Minimization[最小化]

- Any DFA can be converted to its minimum-state equivalent DFA
  - Discover sets of equivalent states
  - Represent each such set with just one state
- Two states are equivalent if and only if:
  - $\forall \alpha \in \Sigma$, transitions on α lead to equivalent states
  - α-transitions to distinct sets ⇒ states must be in distinct sets



Initial: {A}, {BC, AC}

Initial sets:
{non-accepting states}, {accepting states}

For {BC, AC}

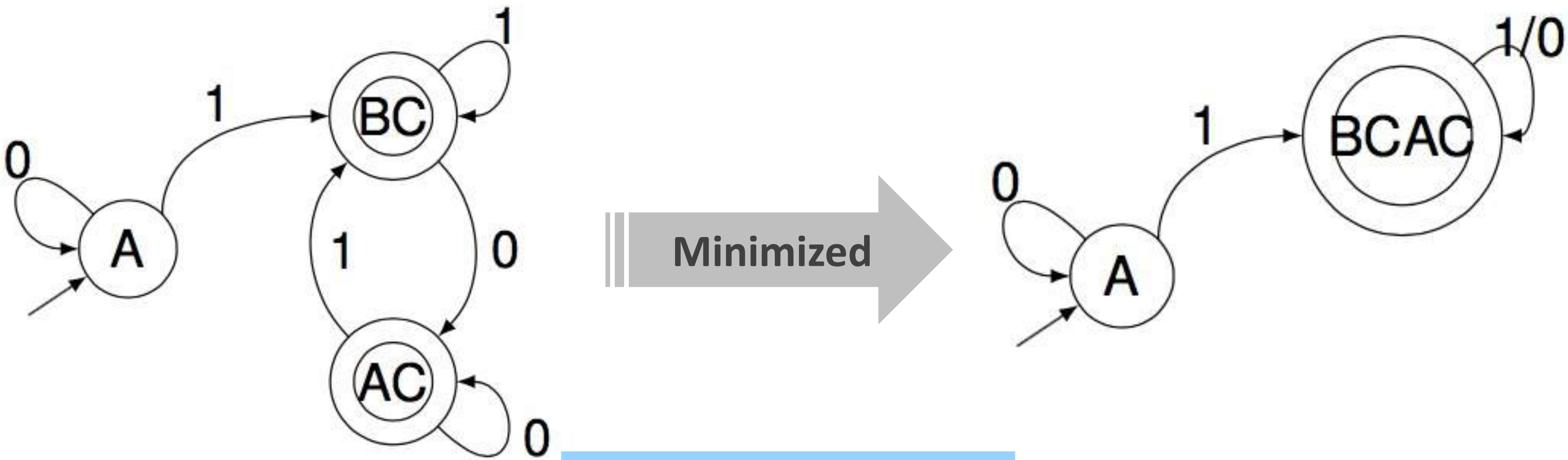
- BC on ‘0’ → AC, AC on ‘0’ → AC
- BC on ‘1’ → BC, AC on ‘1’ → BC
- No way to distinguish BC from AC on any string starting with ‘0’ or ‘1’

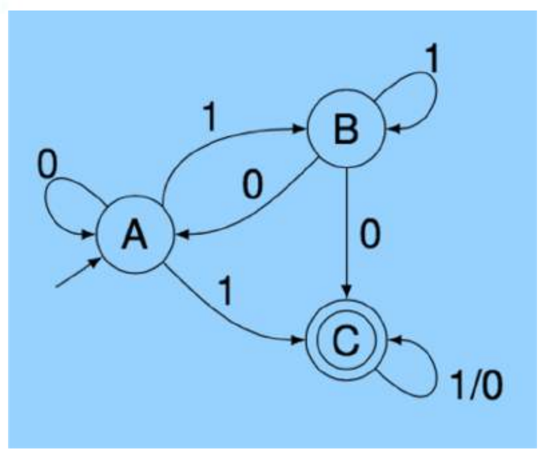
Final: {A}, {BCAC}

https://people.cs.umass.edu/~moss/610-slides/06.pdf

NFA → DFA: Minimization (cont.)

- States *BC* and *AC* do not need differentiation
  - Should be merged into one





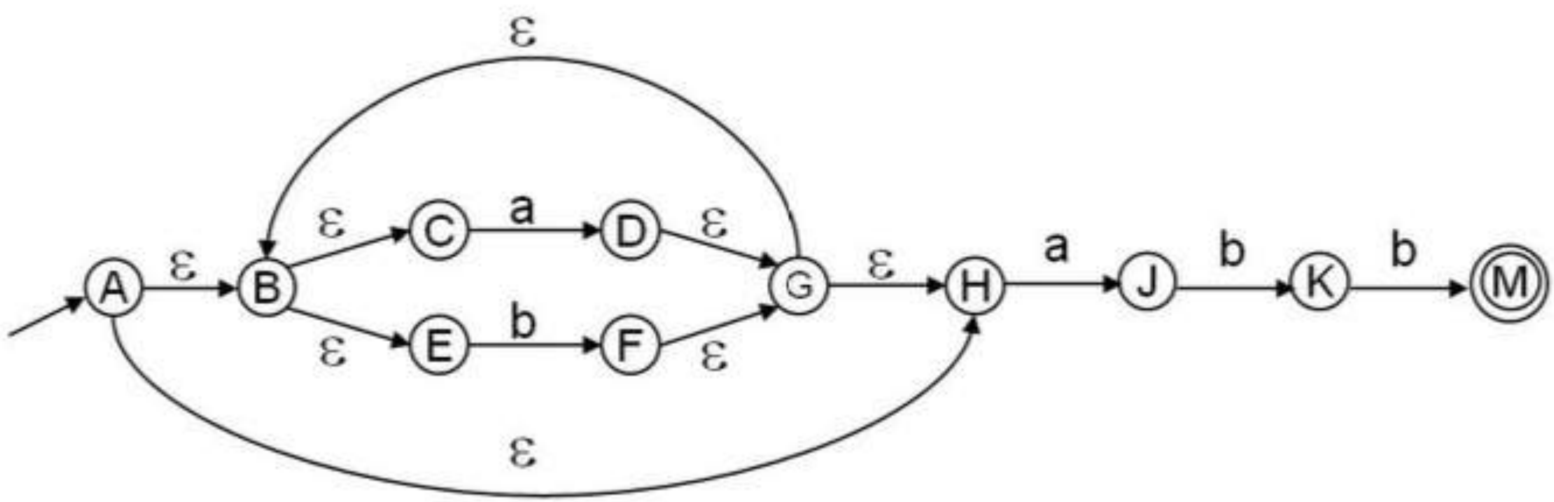
Minimization Algorithm

- The algorithm
  - Partitioning the states of a DFA into groups of states that cannot be distinguished (i.e., equivalent)
  - Each groups of states is then merged into a single state of the min-state DFA
- For a DFA （$\sum$, S, n, F, δ）
  - The initial partition P_0, has two sets {F} and {S-F}
  - Splitting a set (i.e., partitioning a set ***s*** by input symbol α)
    - Assume q_a and $q_b \in$ ***s***, and $\delta(q_a, \alpha) = q_x$ and $\delta(q_b, \alpha) = q_y$
    - If q_x and q_y are not in the same set, then ***s*** must be split (i.e., α splits ***s***)
    - One state in the final DFA cannot have two transitions on α

```
P <- {F}, {S-F}
while (P is still changing)
  T <- { }
   for each state s ∈ P
      for each α ∈ Σ
         partition s by α into s1 and s2
         T <- T ∪ s1 ∪ s2
   if T ≠ P then
      P <- T
```

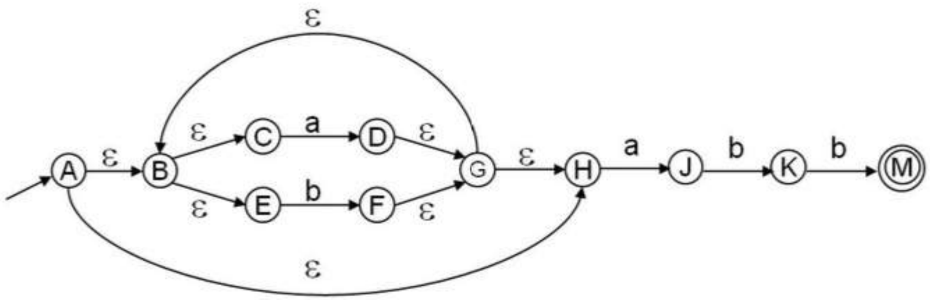

https://people.cs.umass.edu/~moss/610-slides/06.pdf

NFA → DFA: More Example



- Start state of the equivalent DFA
  - ε-closure(A) = {A, B, C, E, H} = A'
- ε-closure(move(A', a)) = ε-closure({D, J}) = {B, C, D, E, H, G, J} = B'
- ε-closure(move(A', b)) = ε-closure({F}) = {B, C, E , F, G, H} = C'
- … …

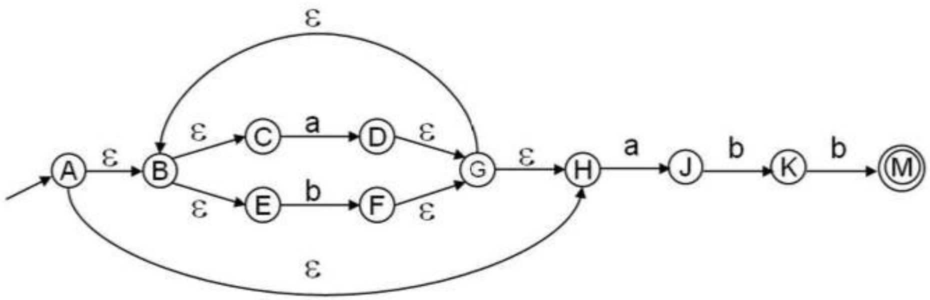
Step 1: Construct the NFA Table



State table of the NFA ⟹

	ε	a	b
A	BH		
B	CE		
C		D	
D	G		
E			F
F	G		
G	BH		
H		J	
I			
J			K
K			M
M			

Step 2: Update ε Column to ε-closure

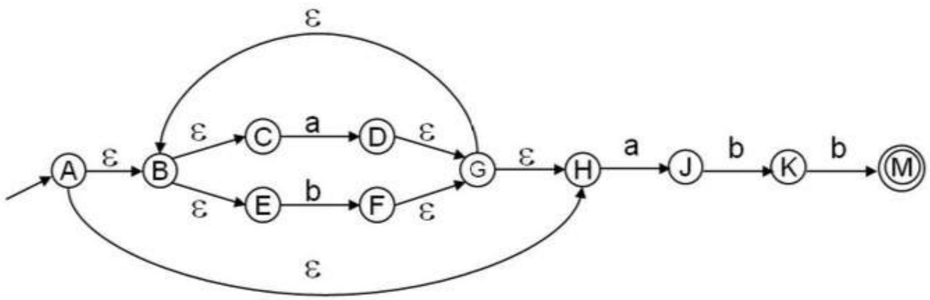


ε-closure of the NFA state

e.g., ε-closure(D) = {D,B,H,C,E} ⟹

	ε	a	b
A	ABHCE		
B	BCE		
C		D	
D	DBHCE		
E			F
F	FGBHCE		
G	GBHCE		
H		J	
I			
J			K
K			M
M			

Step 3: Update other cols based on the ε col



Get the transitions of the ε col
e.g., {D,B,H,C,E} --a--> {D,J}

⟹

	ε	a	b
A	ABHCE	DJ	F
B	BCE	D	F
C		D	
D	DBHCE	DJ	F
E			F
F	FGBHCE	DJ	F
G	GBHCE	DJ	F
H		J	
I			
J			K
K			M
M			

Step 4: Construct the DFA Table

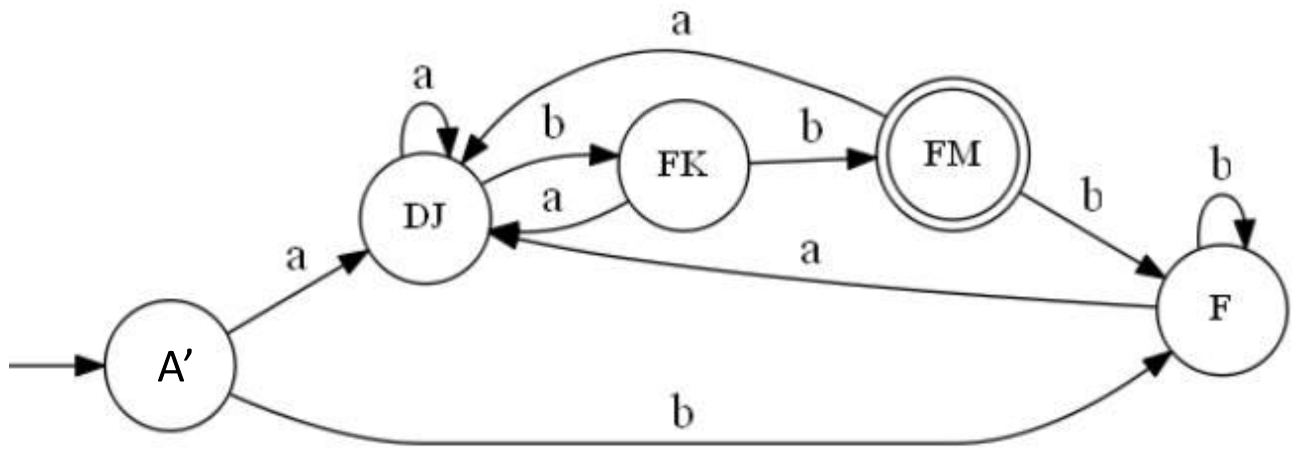
	a	b
A’	DJ	F
DJ	DJ	FK
F	DJ	F
FK	DJ	FM
FM	DJ	F

	ε	**a**	**b**
A	ABHCE A’ for short	DJ	F
B	BCE	D	F
C		D	
D	DBHCE	DJ	F
E			F
F	FGBHCE	DJ	F
G	GBHCE	DJ	F
H		J	
I			
J			K
K			M
M			

Step 4: Construct the DFA Table(cont.)

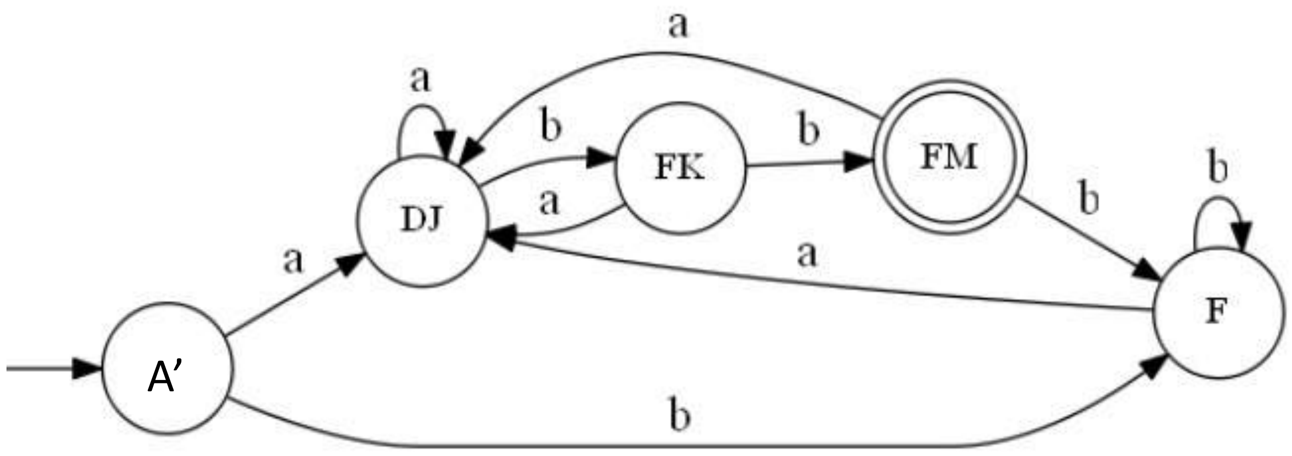
	a	b
A’	DJ	F
DJ	DJ	FK
F	DJ	F
FK	DJ	FM
FM	DJ	F

- Is the DFA minimal?
  - States A’ and F should be merged
- Should we merge states A and FM?
  - NO. A’ and FM are in different sets from the very beginning (FM is t).

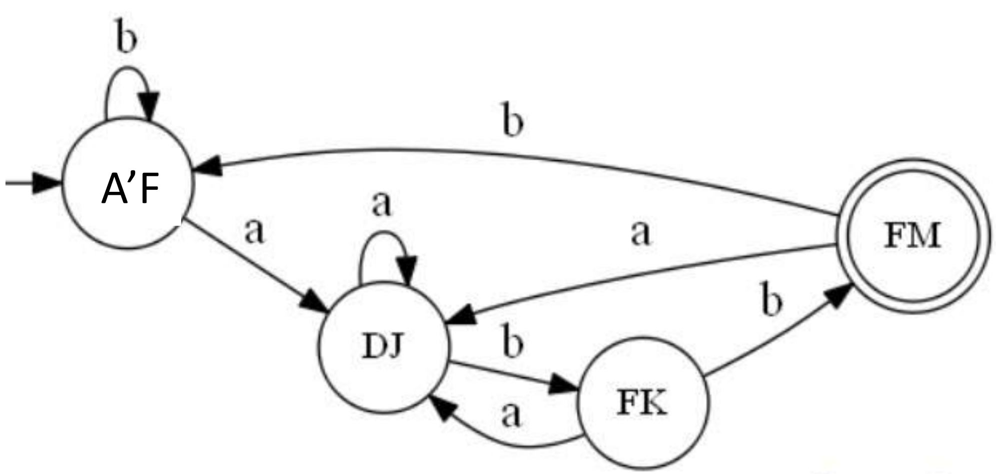


Step 5: (Optional) Minimize DFA

- Original DFA: before merging A' and F



- Minimized DFA: Do you see the original RE (a|b)*abb



NFA → DFA: Space Complexity[空间复杂度]

- NFA may be in many states at any time

- How many different possible states in DFA?
  - If there are *N* states in NFA, the DFA must be in some subset of those *N* states
  - How many non-empty subsets are there?
    - 2^N-1

- The resulting DFA has $O(2^N)$ space complexity, where *N* is number of original states in NFA
  - For real languages, the NFA and DFA have about same number of states

NFA → DFA: Time Complexity[时间复杂度]

- DFA execution
  - Requires O(|X|) steps, where |X| is the input length
  - Each step takes constant time
    - If current state is S and input is c, then read T[S, c]
    - Update current state to state T[S, c]
  - Time complexity = O(|X|)

Deterministic: unique transition

- NFA execution
  - Requires O(|X|) steps, where |X| is the input length
    - Anyway, the input symbols should be completely processed
  - Each step takes $O(N^2)$ time, where *N* is the number of states
    - Current state is a set of potential states, up to *N*
    - On input c, must union all $T[S_{potential}, c]$, up to *N* times
      - Each union operation takes O(N) time
  - Time complexity = $O(|X|*N^2)$

Non-deterministic: form current state, your can transit to any (including itself)