

언어 인식기 (오토마타)

올바른 문법의 언어가 들어왔는지 체크해준다

- Turing machine
- Linear Bounded Automata
- Pushdown Automata
- Finite Automata (정규언어)

유한 오토마타의 정의

Q : state들의 집합

Σ : input alphabet 들의 집합

δ : mapping 함수

q_0 : start state (시작)

F : Final state의 집합 (여러개의 종료상태가 있을 수 있다)

δ : 문법에서의 생성규칙 같은 것

state 간의 이동을 표현

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

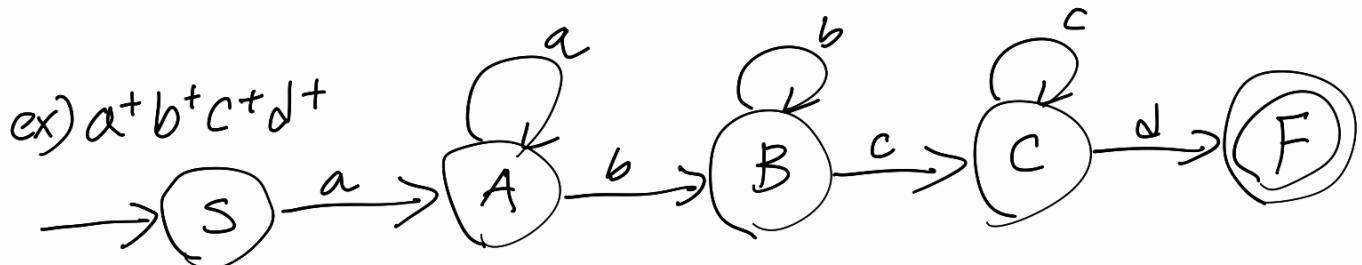
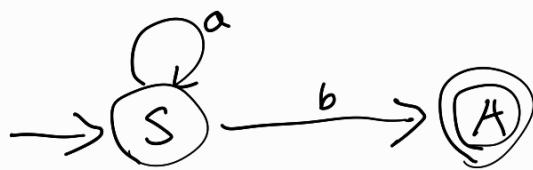
$$G = [V_n, V_T, P, S]$$

$$re: \emptyset, \epsilon, a, +, *, *$$

$$M = (Q, \Sigma, \delta, q_0, F)$$

유한 오토마타

ex) $a^* b$



$Q: S, A, B, C, F$

$\Sigma: a, b, c, d$

$\varepsilon: 각 state의 화살표 갯수 (7개)$

$q_0: S$

$F: F$

〈상태 전이표〉

	a	b	c	d
S	A			
A	A	B		
B		B	C	
C			C	F
F				

* 프로그램을 구현할때는 상태전이표로 표현하는게 훨씬 편하다.

Final state 까지 올수 있는 모든 string들의 집합이
이 오토마타가 인식하는 정규언어이다.

* 언어의 string을 생성하는건 "문법"
 string이 그 언어에 속하는지 판단하는 프로그램은 "오토마타"

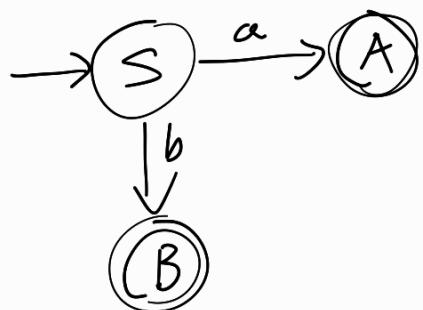
- String을 다 처리했는데 final state에 없으면 Yes!
- String을 다 처리했는데 final state에 있지 않거나
 String을 전부 인식하지 못하면 No!

식별자 (Identifier) 오토마타



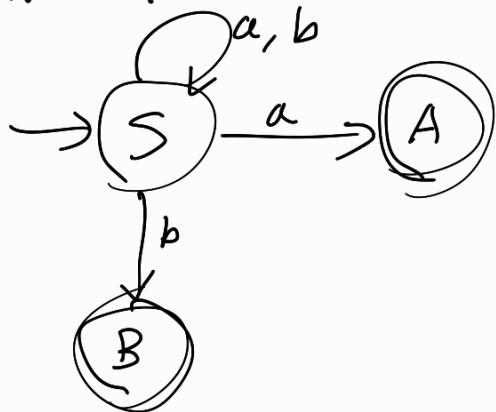
결정적 유한 오토마타 (DFA)

각 state에서 입력 심볼에 대해 next state가 항상
 1개로 결정



비결정적 유한 오토마타 (NFA)

입력 심볼에 대해 next state가 1개로 결정되지 않는 경우



- DFA는 $O(n)$ 시간에 해당 언어를 인식하는 프로그램을 쉽게 구현 가능
 - if문 : 오토마타가 달라지면 프로그램 다시 짜야함
 - 2차원 테이블만 바꿔주면 됨
- NFA는 백트래킹을 쓰는 등 구현이 어렵고, 설령 구현해도 시간복잡도가 크게 둘다
 \therefore , NFA를 DFA로 변환시켜 주야 한다

NFA의 예)

$[A, B]$ state 같은 경우
A와 B가 가리키는 것을
함께 state를 가리
키고 있다

$[A, B]$ $[B, C, D]$

	a	b
S	$\{A\}$	$\{B, C\}$
A	$\{A, B\}$	$\{B, D\}$
B		$\{B, D\}$
C	$\{D\}$	
D		$\{C\}$

'A 또는 B' 같은 경우를
하나의 unique한 state로
 $[A, B]$ 로 만들어 줘야 한다

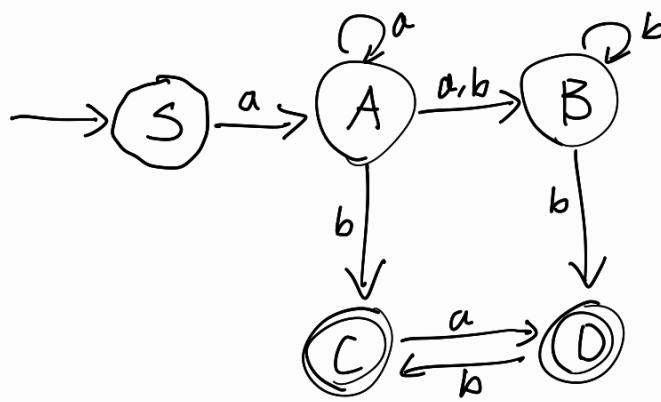
Final state = $\{C, D\}$

C와 D가 포함된 모든 state는
Final state이다.

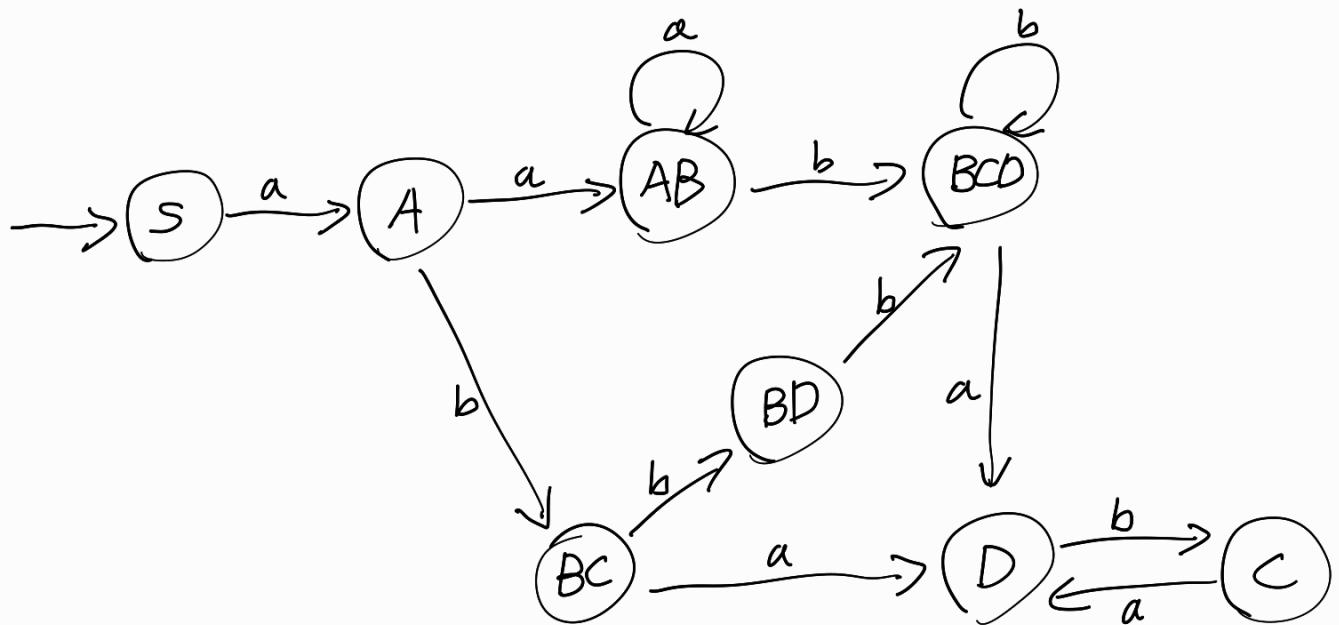


	a	b
$\{S\}$	$\{A\}$	
$\{A\}$	$\{A, B\}$	$\{BC\}$
$\{A, B\}$	$\{A, B\}$	$\{B, C, D\}$
$\{B, C\}$	$\{D\}$	$\{B, D\}$
$\{B, C, D\}$		$\{B, C, D\}$
$\{D\}$		$\{C\}$
$\{B, D\}$		$\{B, C, D\}$
$\{C\}$	$\{D\}$	

Final state = $\{[B, C], [B, C, D], [D], [B, D], [C]\}$



↓

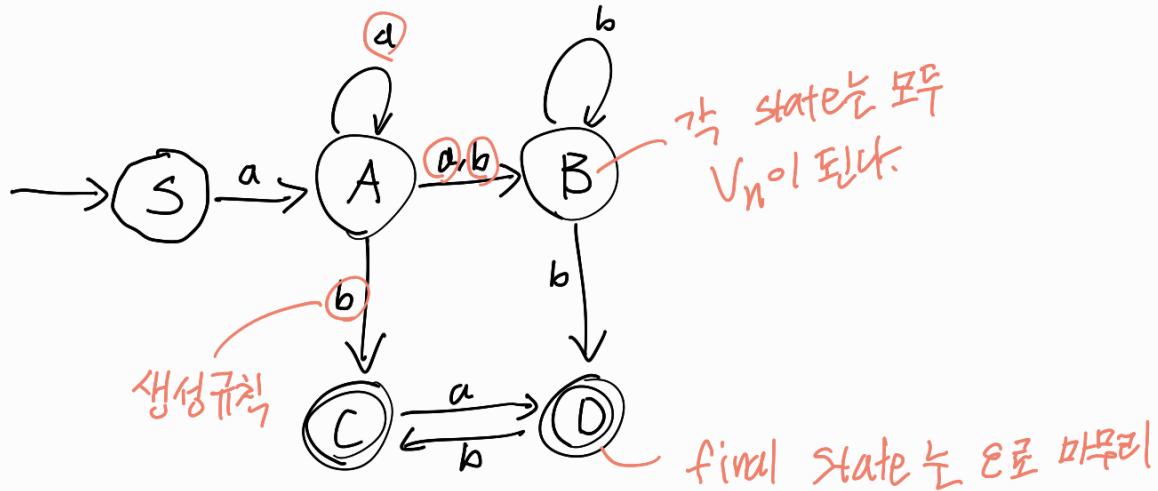


NFA가 DFA로 변환될 때 최대 State 수는

각 NFA의 state를 조합하는 모든 경우의 수에서 1을 뺀
경우이므로

$$2^n - 1 \text{ 이다.}$$

유한 오토마타 \rightarrow 정규문법



$$S \rightarrow aA$$

$$A \rightarrow aA \mid aB \mid bB \mid bC$$

$$B \rightarrow bB \mid bD$$

$$C \rightarrow aD \mid \epsilon$$

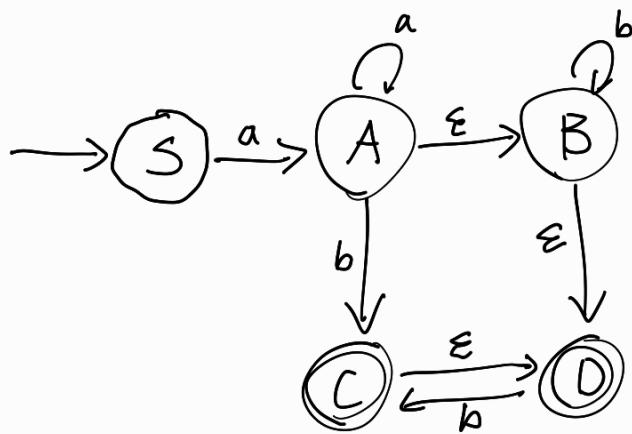
$$D \rightarrow bC \mid \epsilon$$

정규문법 \rightarrow 유한 오토마타

그냥 하면 된다

이제 어떤 정규문법이든지 우상향이면 finite automata로
기술할 수 있다

ϵ -NFA



- ϵ 은 input 없이 전이 되는 것을 말한다
- ϵ -CLOUSER 함수는 ϵ -transition으로 도달 가능한 state의 집합입니다.

$$\epsilon\text{-CLOUSER}(A) = \{A, B, D\}$$

$$\epsilon\text{-CLOUSER}(B) = \{B, D\}$$

$$\epsilon\text{-CLOUSER}(C) = \{C, D\}$$

ϵ -NFA를 DFA로 변환

	a
S	$\{A\}$

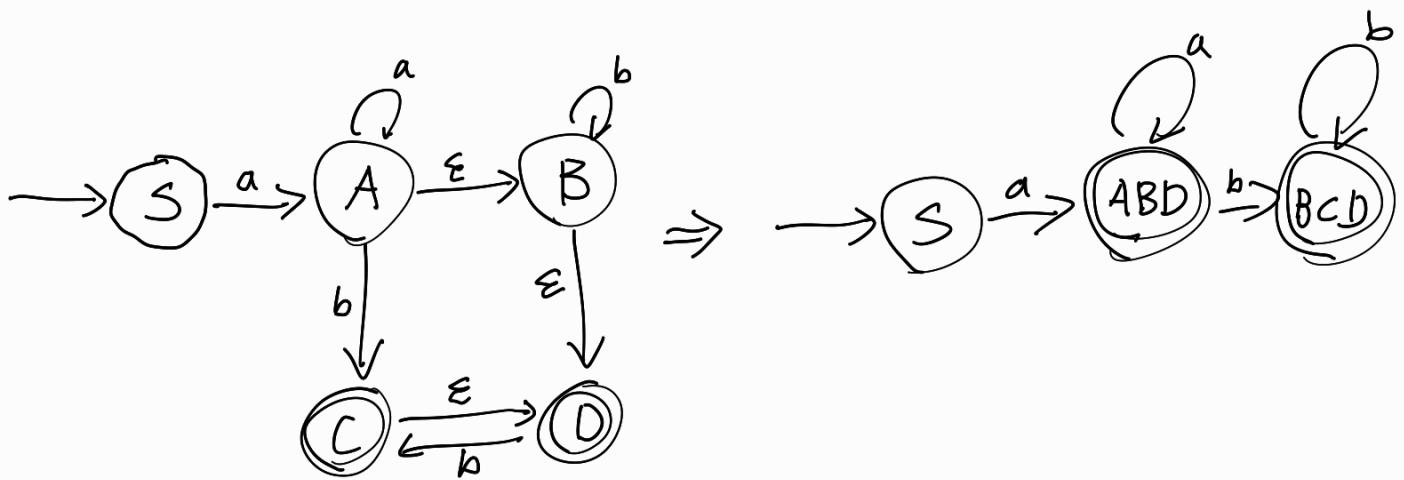
 \rightarrow

	a
S	$\{ABD\}$

	a	b
$[S]$	$[A, B, D]$	
$[A, B, D]$	$[A, B, D]$	$[B, C, D]$
$[B, C, D]$	$[A, B, D]$	$[B, C, D]$

위에서와 같이 C, D 포함된 것

$$\text{Final State} = \{[A, B, D], [B, C, D]\}$$



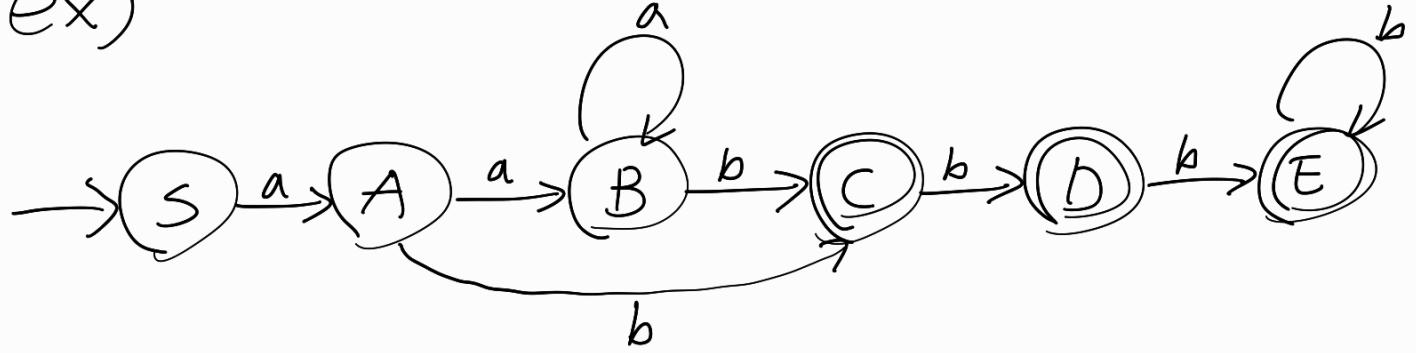
가장 효율적인 DFA로 변화

DFA를 최소화 하는 방법

1단계. final끼리 통합하고 nonfinal state끼리 통합해서
2개 state의 DFA를 만든다

2단계. 이때 DFA로 통합될수 있는 state가 있으면
분할한다.

ex)



	a	b
S	A	C
A	B	C
B	B	D
C		E
D		E
E		E

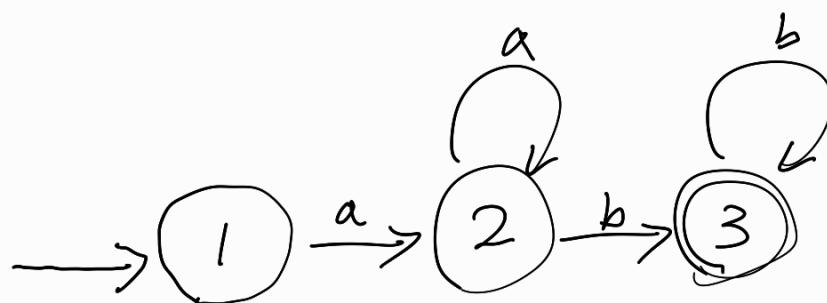
Final state $\{C, D, E\}$

1단계) nonfinal, final 2개로 분할

	1: $\{S, A, B\}$	2: $\{C, D, E\}$
a	1 1 1	$\emptyset \emptyset \emptyset$
b	$\emptyset 2 2$	2 2 2

2단계) DFA 요건 불만족 State 분할

	1: $\{S\}$	2: $\{A, B\}$	3: $\{C, D, E\}$
a	2	2 2	$\emptyset \emptyset \emptyset$
b	\emptyset	3 3	3 3 3

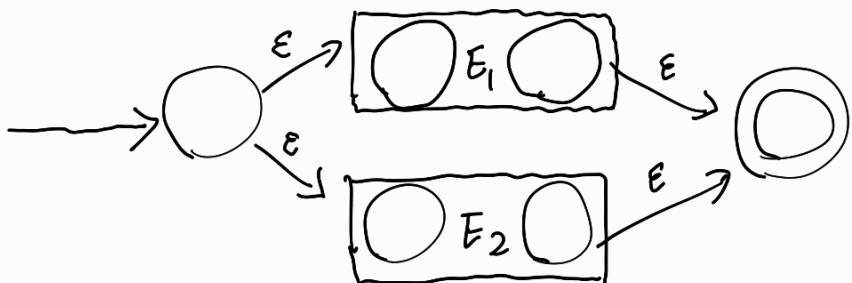


정규문법 \rightarrow 유한 오토마타 위에서 배움

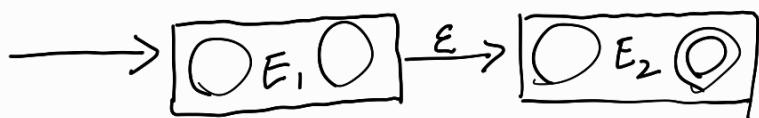
정규표현은?

↳ 정규표현 \rightarrow 유한 오토마타

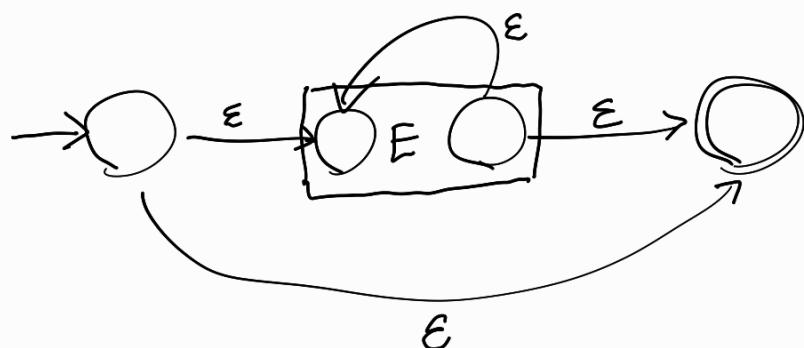
정규표현 $E_1 + E_2$ 를 인식하는 유한 오토마타



정규표현 $E_1 \cdot E_2$ 를 인식하는 유한 오토마타



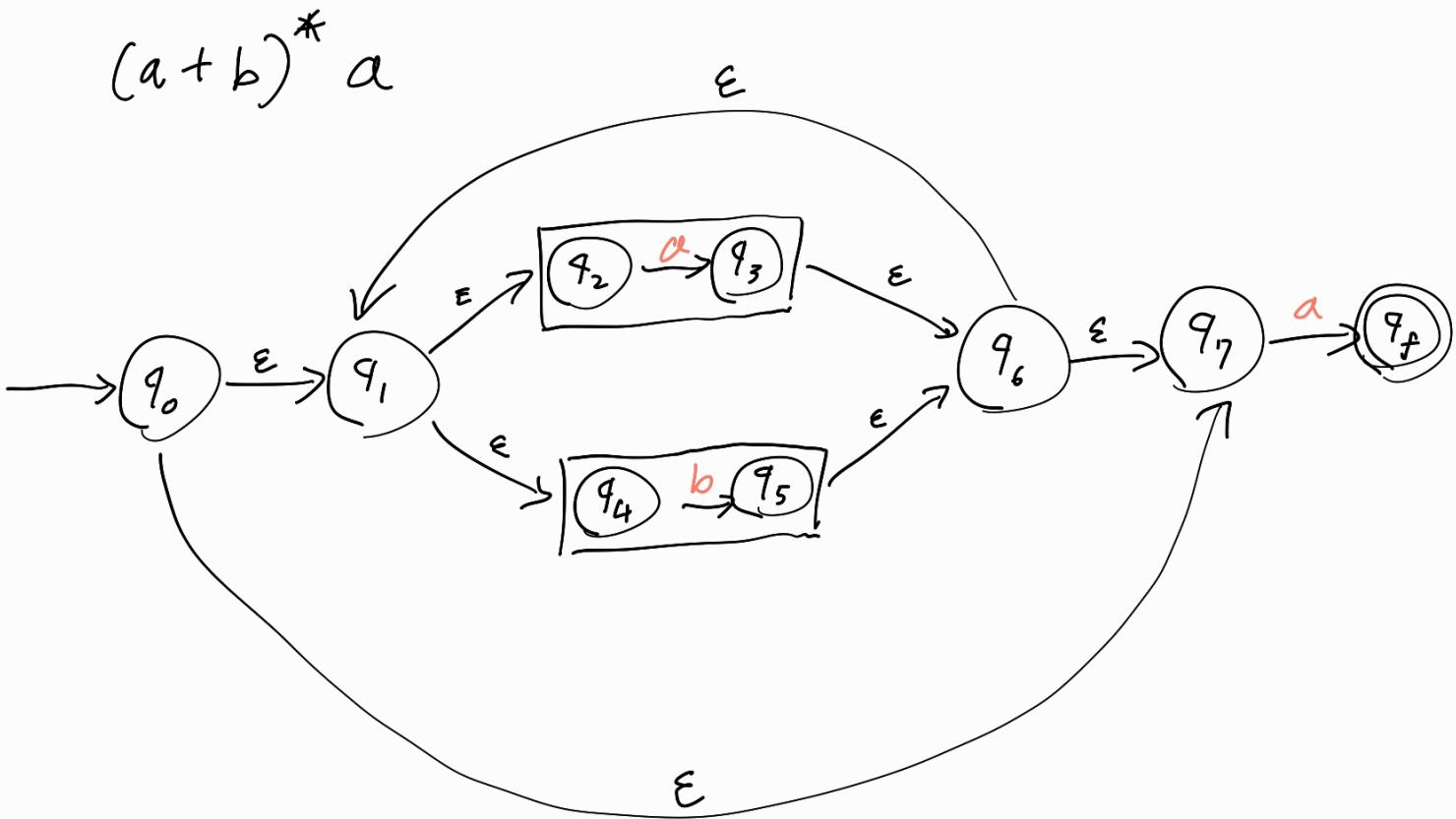
정규표현 E^* 를 인식하는 유한오토마타



이 3가지만 있으면 정규표현을 모두 ϵ -NFA로 나타낼 수 있다

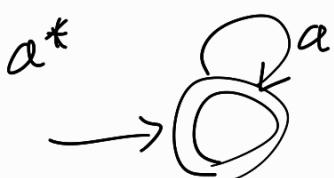
이는 위에서 배운대로 DFA로 변환할 수 있다

ex)



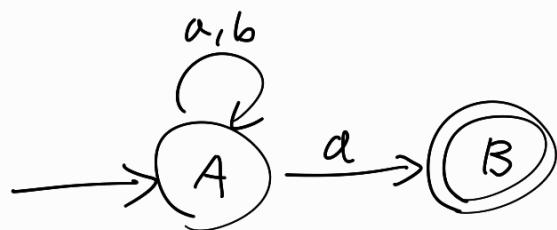
간소화된 유한 오토마타

정규표현식 E, E_1, E_2, \dots 가 terminal이면 위처럼 ϵ 으로 복잡하게 나타낼 필요가 없다



ex)

- $(a+b)^*a$ 에 대한 간소화된 유한 오토마타



- 정규문법

$$A \rightarrow aA \mid bA \mid aB$$

$$B \rightarrow \epsilon$$

- DFA로 변환

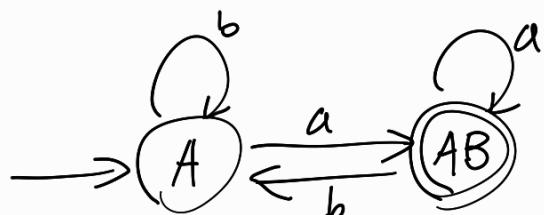
	a	b
A	A, B	A
B		

$$\text{Final state} = \{B\}$$

\Rightarrow

	a	b
[A]	[A, B]	[A]
[A, B]	[A, B]	[A]

$$\text{Final state} = \{[A, B]\}$$



보다시피 NFA를 안 거치고 바로 DFA로 만들어 버리기에는 너무 복잡해서 어렵다

Pumping Lemma

- 어떤 언어가 정규언어가 아님을 증명하는데 활용
- 유한 오토마타의 유한개의 state 보다 긴 string을 인식하려면 state에서 반복되는 부분이 있어야 한다

state보다 긴 string $w = \alpha\beta\gamma$ 의 반복되는 부분을 β 라 하면 $\alpha\beta^*y$ 도 이 오토마타를 인식할 수 있다

\therefore 위 조건일 때 $w = \alpha\beta\gamma$ 와 $\alpha\beta^*y$ 를 둘다 만족하지 못한다면 정규언어가 아니다