Lexical Specification

① at least one: $A^+ \equiv AA^*$
② Union: $A|B \equiv A+B$
③ Option: $A? \equiv A+\varepsilon$
④ Range: $'a' + 'b' + \dots + 'z' \equiv [a-z]$
⑤ Excluded range:
complement of $[a-z] \equiv [\hat{}a-z]$

Based on $s \in L(R)$, we need some extension!

1. Write a rexp for the lexemes of each token class.
- Number = digit$^+$
- keyword = 'if' + 'else' + ...
- Identifier = letter(letter + digit)*
- OpenPar = '('
- ...

2. Construct R, matching all lexemes for all tokens.

R = keyword + Identifier + Number + ...
$= R_1 + R_2 + \cdots$

3. Let input be $x_1 \cdots x_n$
For $1 \le i \le n$ check
$x_1 \cdots x_i \in L(R)$

4. If success, then we know that
$x_1 \cdots x_i \in L(R_j)$ for some j

5. Remove $x_1 \cdots x_i$ from input and go to (3)

Regular expressions are a concise notation for string patterns.

Use in lexical analysis requires small extensions

- To solve ambiguities - [matches as long as possible highest priority match]
- To handle errors - put them last and lowest priority

Good algorithms known

- Require only single pass over the input
- Few operations per character (table lookup)

Finite Automata

Regular Expression = Specification
Finite Automata = Implementation

A finite automaton consists of
- An input alphabet: Σ
- A finite set of states: S
- A start state: n
- A set of accepting states: $F \subseteq S$
- A set of transitions: state $\xrightarrow{\text{input}}$ state

Transition

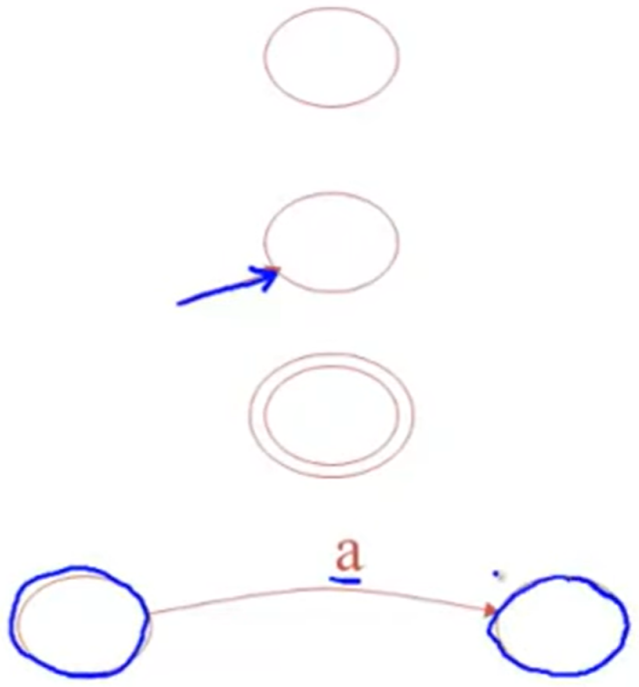
$s_1 \xrightarrow{a} s_2$

Is read In state s_1 on input a go to state s_2

If end of input and in accepting state $\Rightarrow$ accept

Otherwise $\Rightarrow$ reject { terminates in state $s \notin F$; gets stuck

- A state
- The start state
- An accepting state
- A transition



Language of a FA ≡ set of accepted strings

- A finite automaton accepting any number of 1's followed by a single 0
- Alphabet: {0,1}

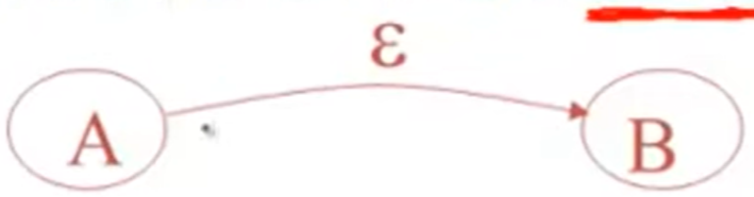
State	Input
A	↑100
A	1↑00
B	10↑0

Reject

State	Input
A	↑110
A	1↑10
A	11↑0
B	110↑

Accepts

- Another kind of transition: ε-moves



State	Input
A	x_1 x_2 x_3
B	x_1 x_2 x_3

Deterministic Finite Automata (DFA)
- One transition per input per state
- No ε-moves

Nondeterministic Finite Automata (NFA)
- Can have multiple transitions for one input in a given state.
- Can have ε-moves.

A DFA takes only one path through the state graph.
An NFA can choose, meaning it can accept if some choices lead to an accepting state.

NFAs and DFAs recognize the same set of languages.

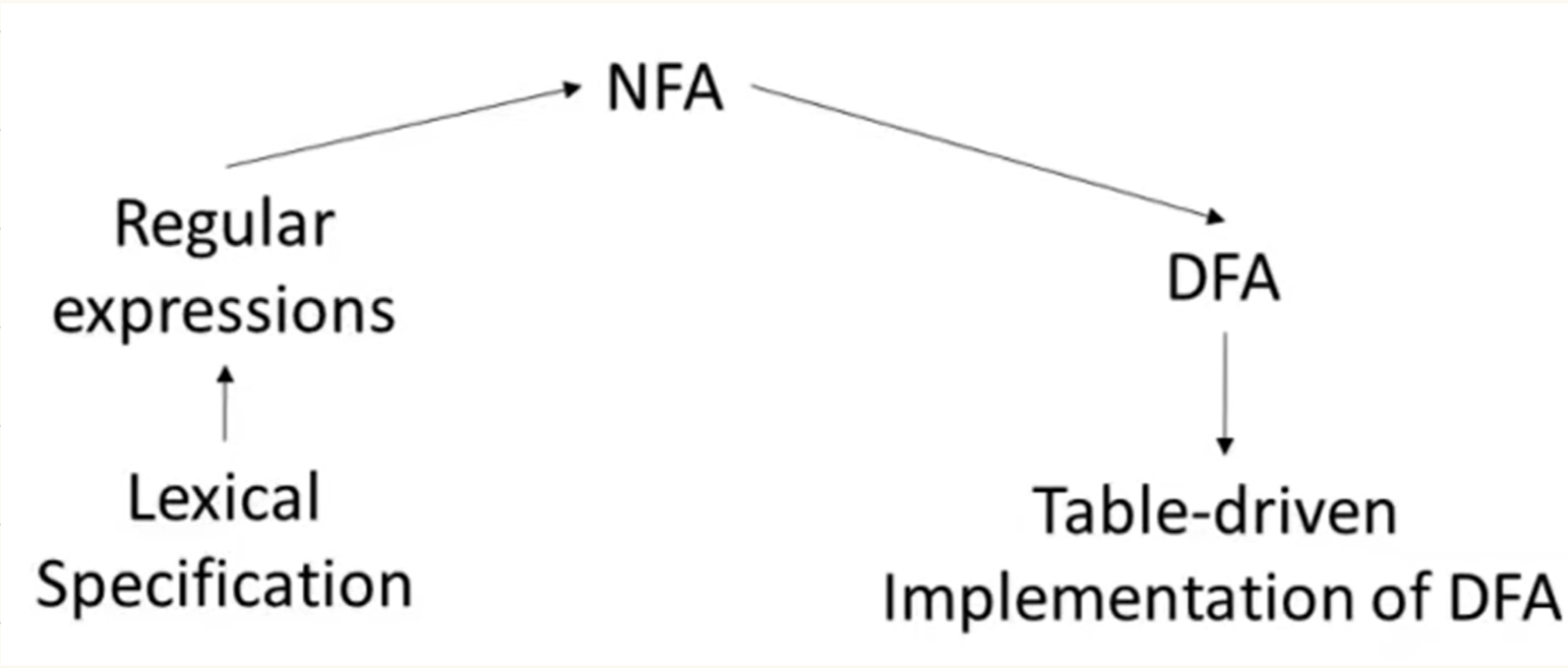
- Regular languages

DFAs are faster to execute

- There are no choice to consider.

NFAs are, in general, smaller.

Regular Expressions to NFAs

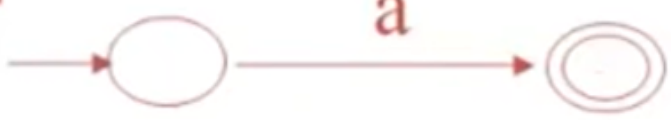


For each kind of rexp, define an equivalent NFA

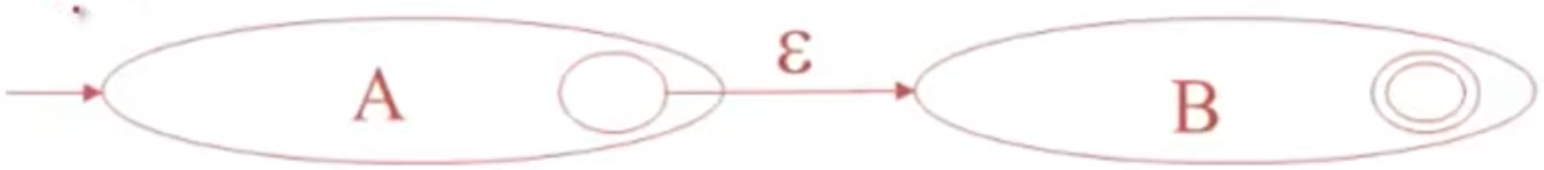
– Notation: NFA for rexp M

- For ε

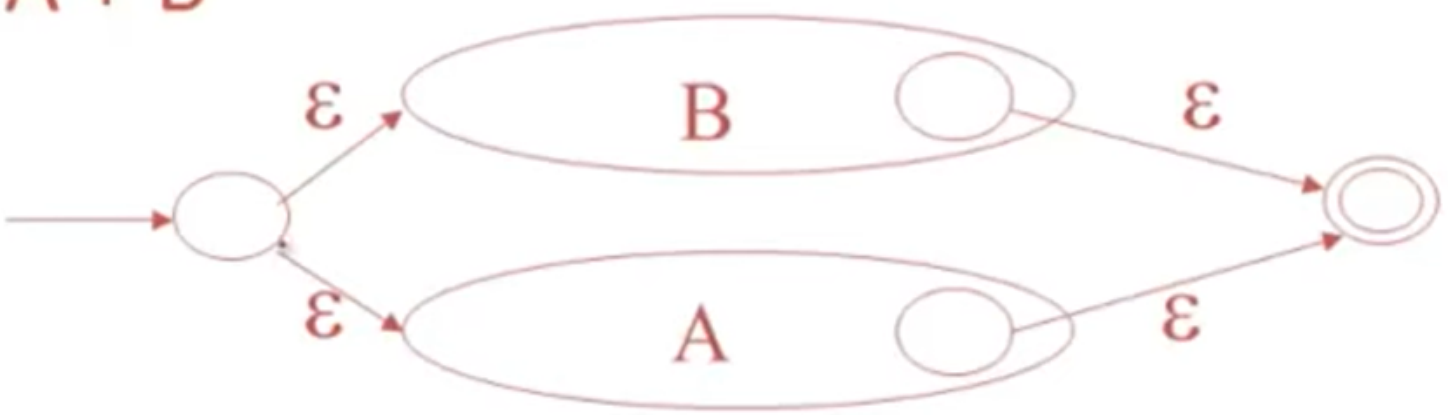
- For input a



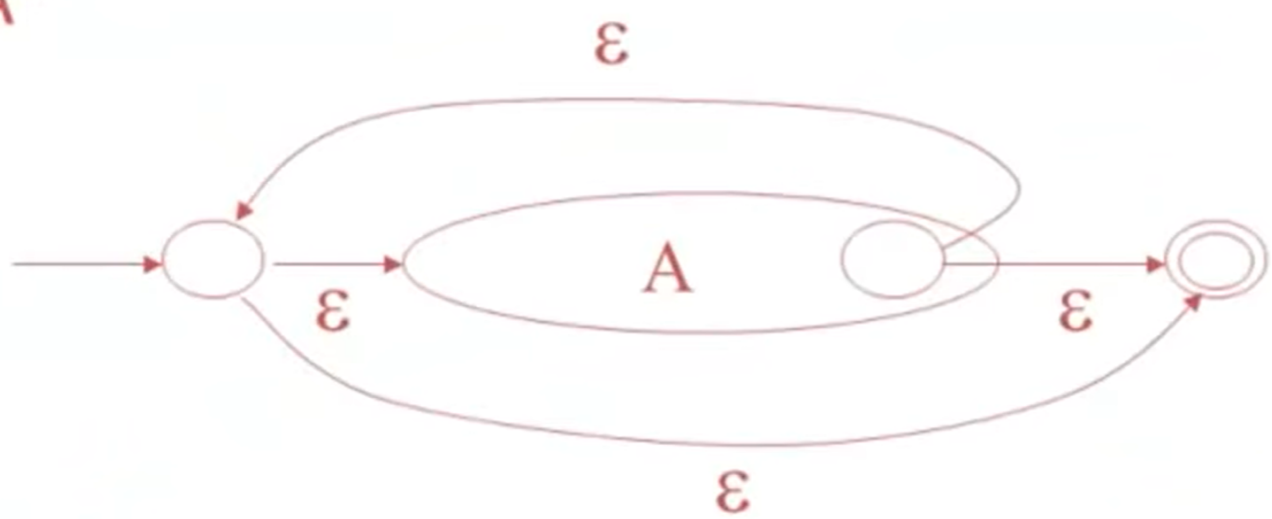
- For AB



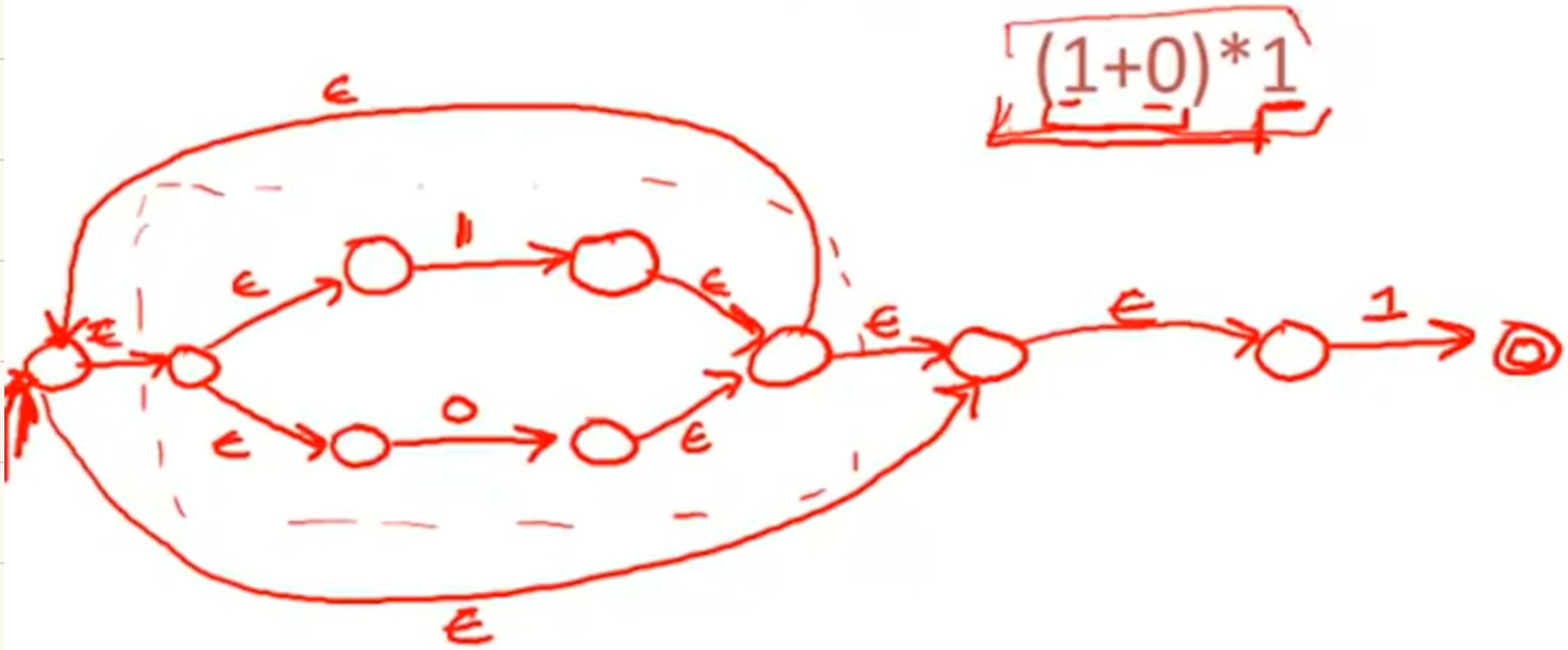
- For A + B



- For A*



- Consider the regular expression



NFA to DFA

An NFA may be in many states at any time.

NFA

states S

start $s \in S$

final $F \in S$

$a(x) = \{y \mid x \in X_n \; x \xrightarrow{a} y\}$

ϵ-clos

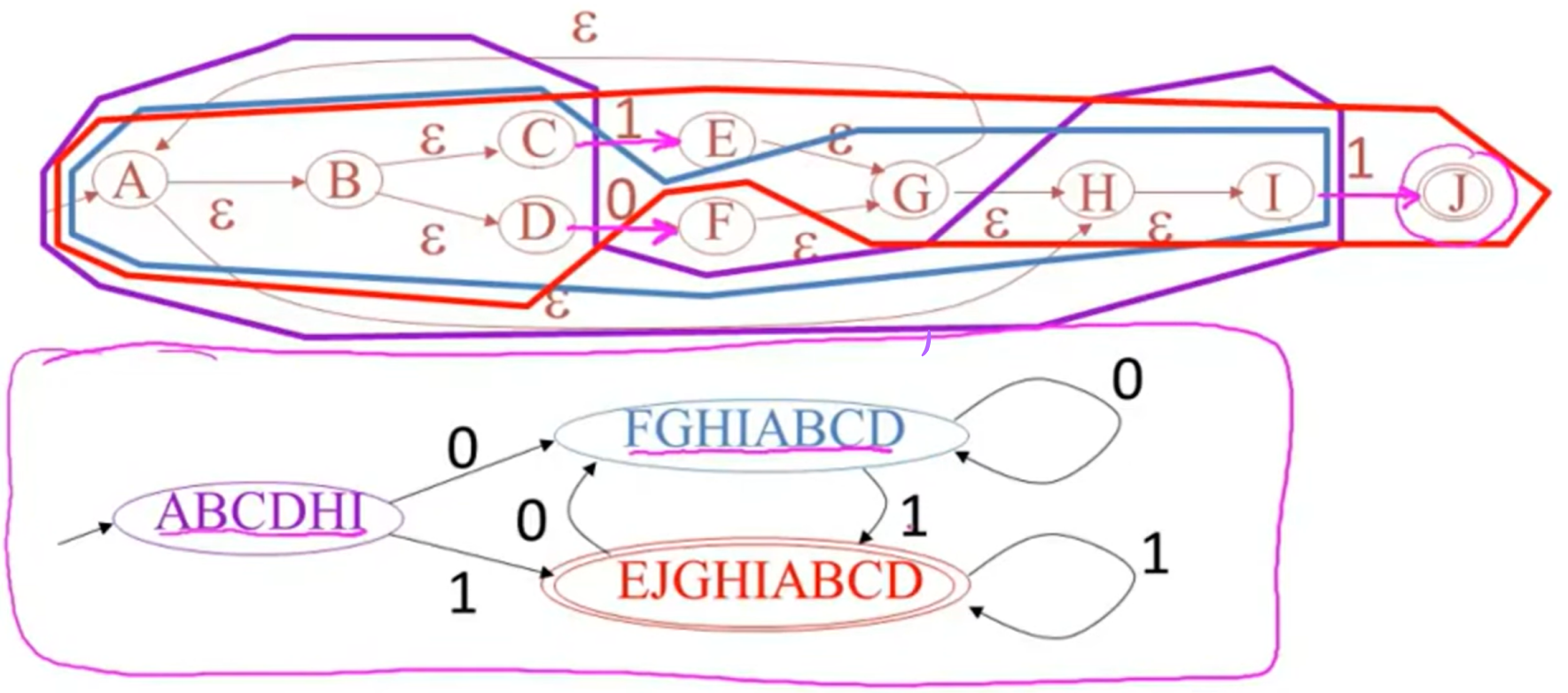
DFA

states subsets of S (except the empty set)

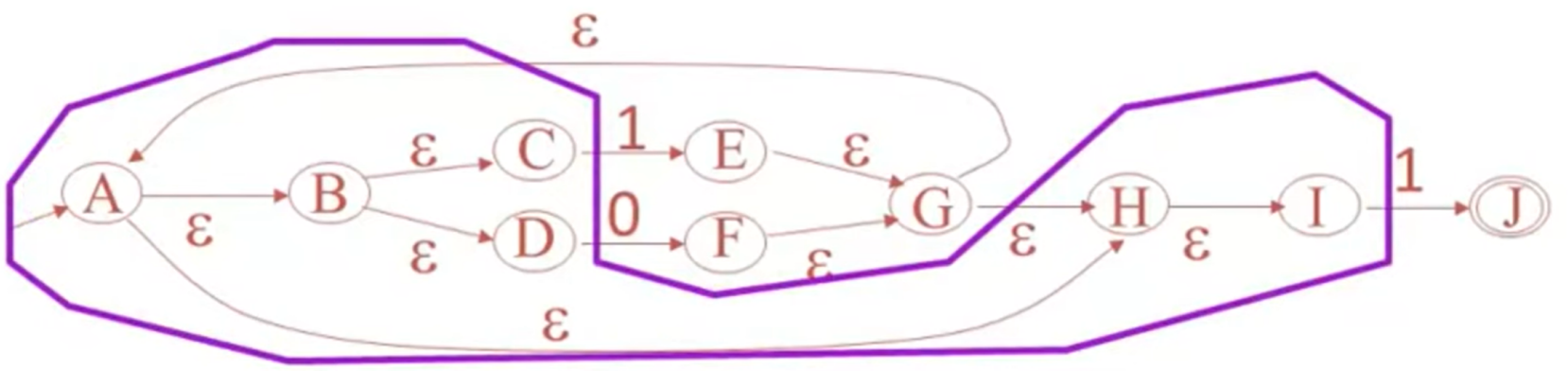
start ϵ-clos(s)

final $\{X \mid X \cap F \neq \phi\}$

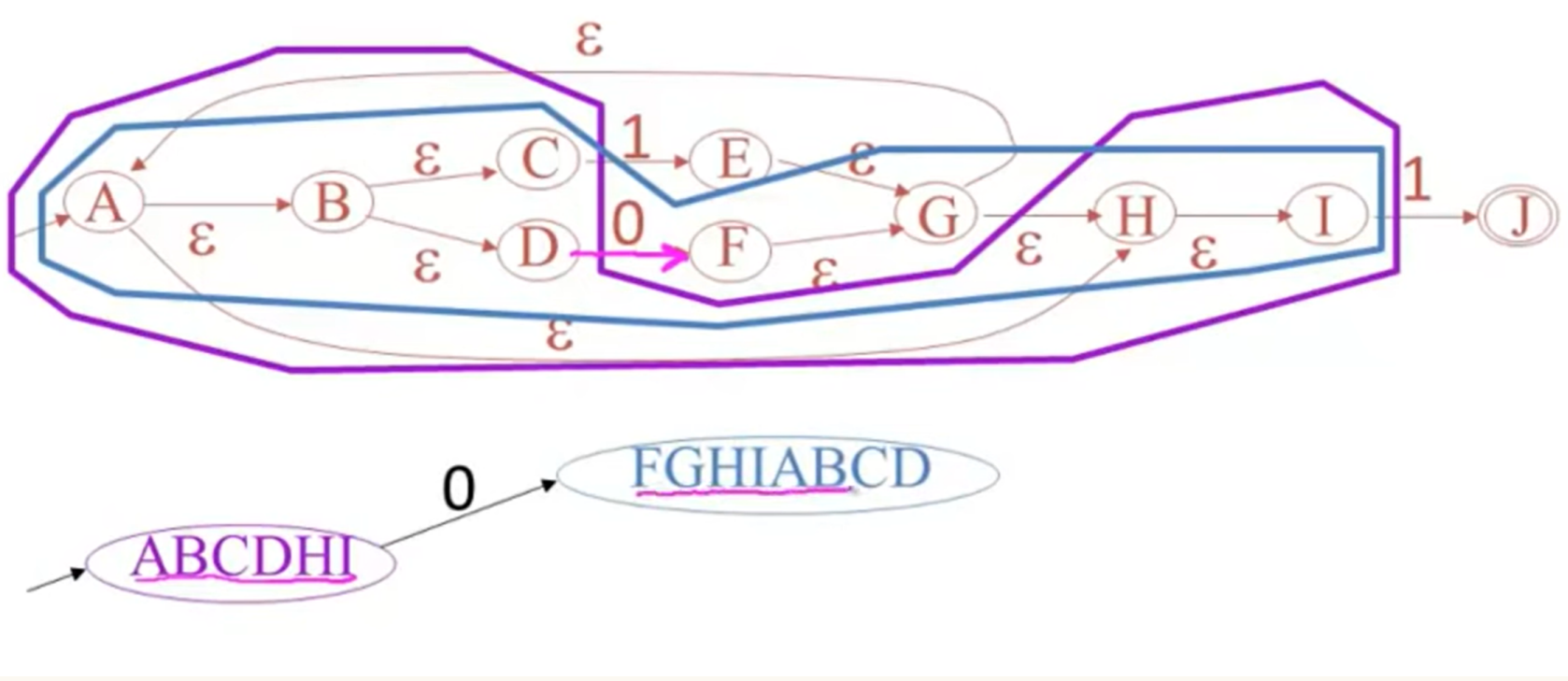
$X \xrightarrow{a} Y$ if $Y = \epsilon$-clos$(a(X))$

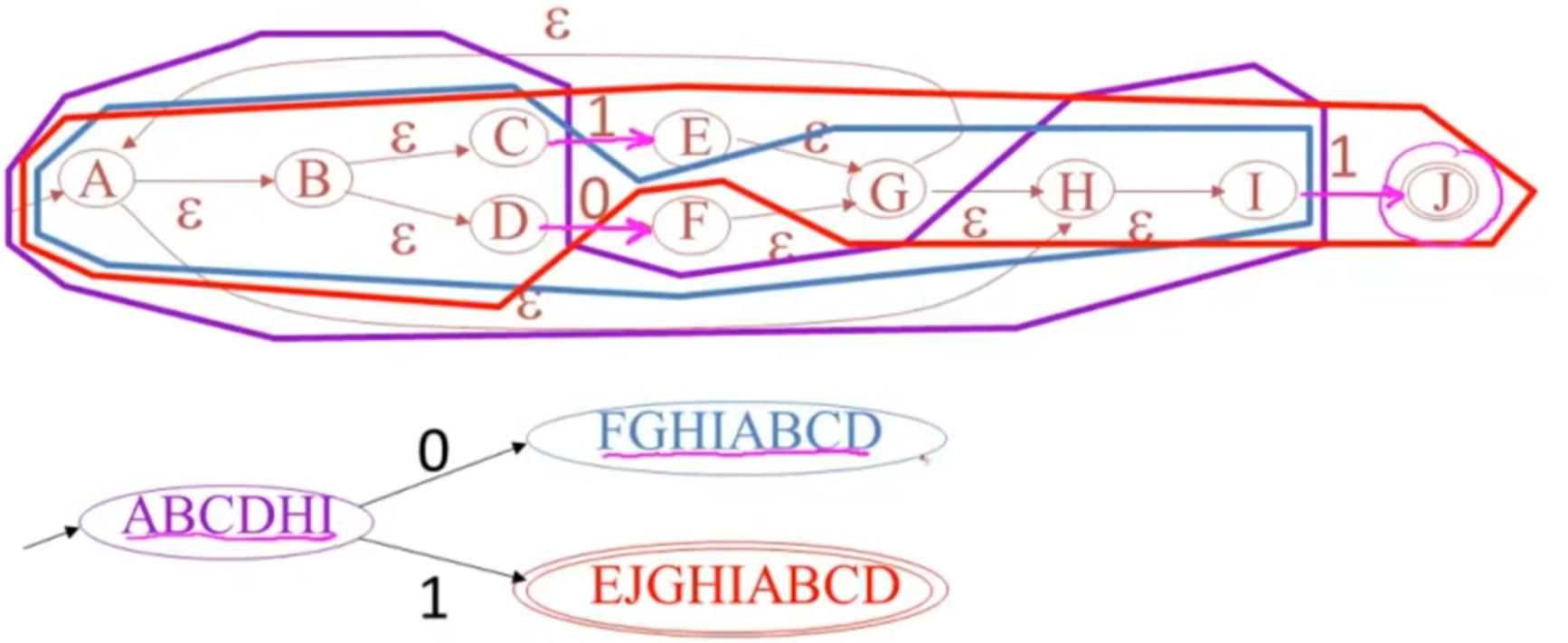


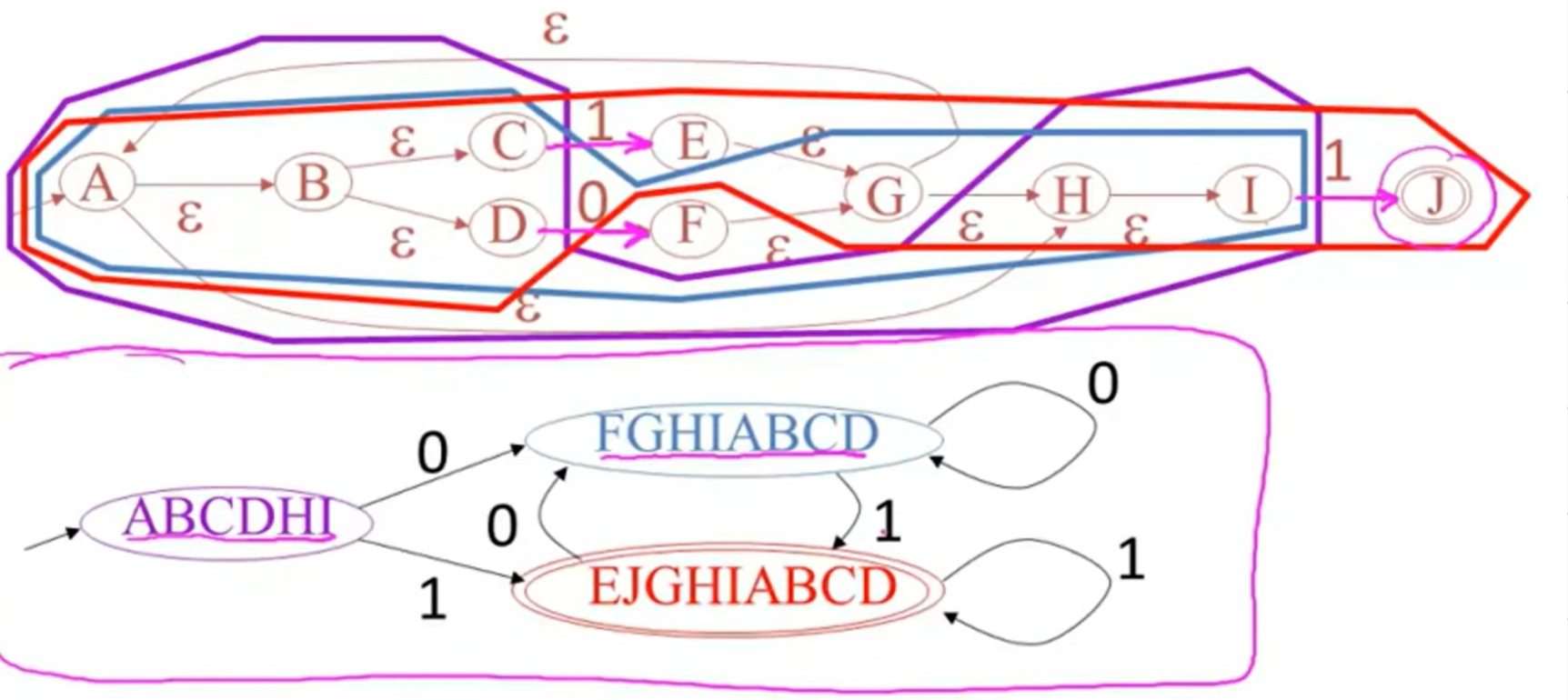
step-by-step analyse



DFA start state
ε-clos(A)

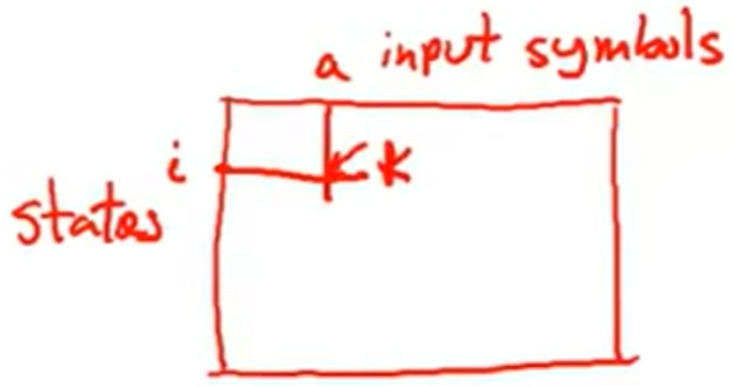


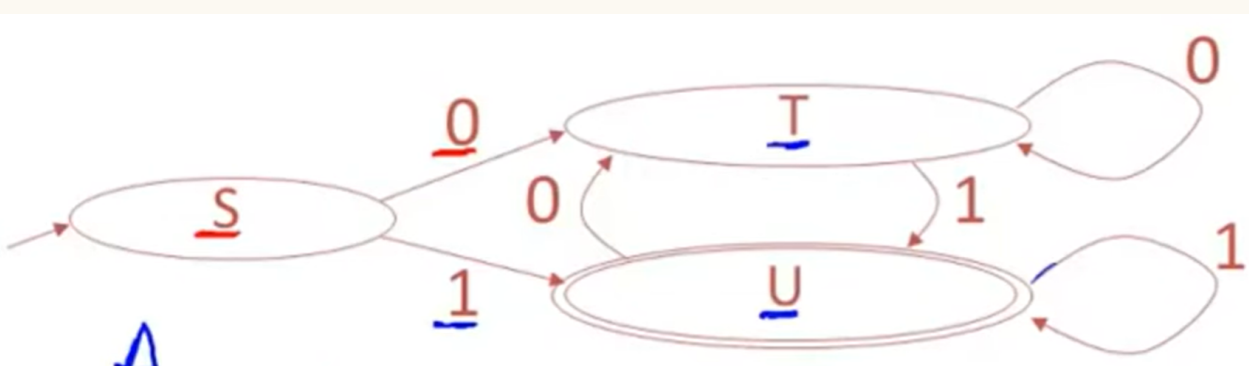

ε
A
B
ε
C
ε
D
ε
1
E
0
F
ε
ε
G
ε
H
ε
I
1
J
ε
0
FGHIABCD
ABCDHI
1
EJGHIABCD


ε
A
B
ε
C
ε
D
ε
1
E
0
F
ε
ε
G
ε
H
ε
I
1
J
ε
0
FGHIABCD
0
ABCDHI
0
1
1
EJGHIABCD
1

Implementing Finite Automata

- A DFA can be implemented by a 2D table T
  - One dimension is states
  - Other dimension is input symbol
  - For every transition $S_i \rightarrow^a S_k$ define T[i,a] = k





A

	0	1
→ S	T	U
T	T	U
U	T	U

```
i = 0;
state = 0;
while (input[i]) {
  state = A[state, input[i++]];
}
```

NFA → DFA conversion is key.

Tools trade between speed and space

DFAs	faster, less compact
NFAs	slower, concise