Lecture 03

Lexical Analysis

Part 2: Recognition of tokens

Hyosu Kim

School of Computer Science and Engineering

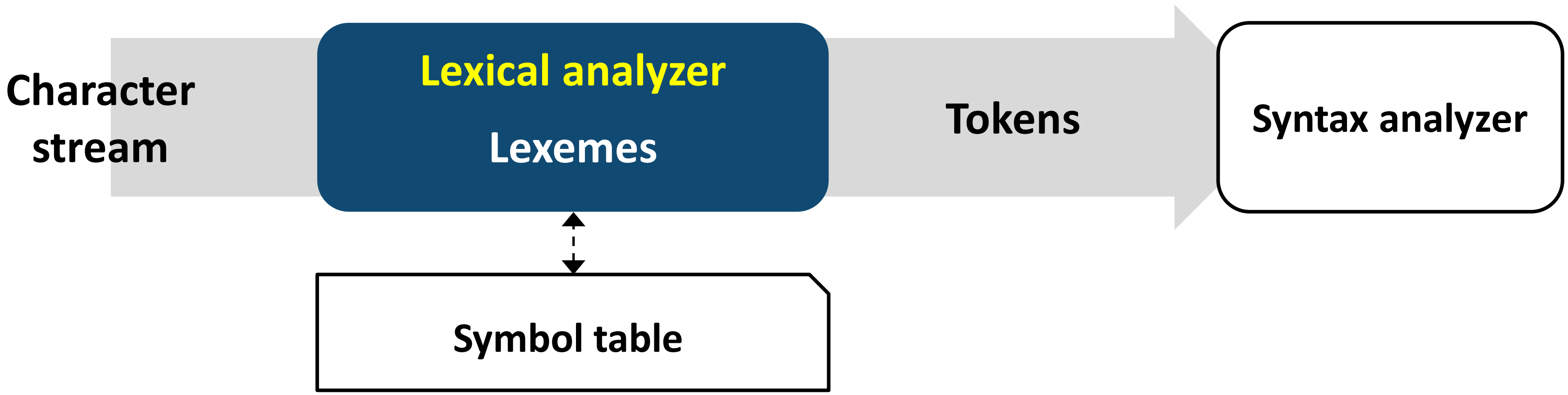
Chung-Ang University, Seoul, Korea

https://hcslab.cau.ac.kr

hskimhello@cau.ac.kr, hskim.hello@gmail.com

Overview

What does a lexical analyzer do?



Remaining questions in designing lexical analyzers

1. How to specify the patterns for tokens? **Regular languages**
2. How to recognize the tokens from input streams? **Finite automata**

Outline

In this lecture, you will learn

1. **What a finite automata is**

2. **How we can recognize tokens with the use of a finite automata**

3. **How to implement a lexical analyzer**

Finite automata

The implementation for recognizing tokens

It accepts or rejects inputs based on the patterns specified in the form of regular expressions

e.g., if $s \in L(token)$, then accept

A finite automata $\boldsymbol{M = \{Q, \Sigma, \delta, q_0, F\}}$

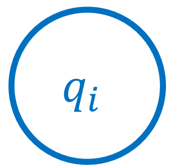
- A finite set of states $Q = \{q_0, q_1, q_2, \ldots, q_i\}$
- An input alphabet Σ = a finite set of input symbols
- A start state q_0
- A set of accepting (or final) states F which is a subset of Q
- A set of state transition functions δ

  e.g., $\delta(q_0, a) = q_1$: the state transition from q_0 to q_1 on the input symbol a

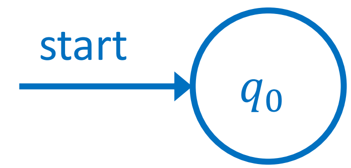
Finite automata

A finite automata can be expressed in the form of graphs, a transition graph

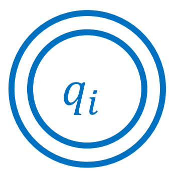
A state



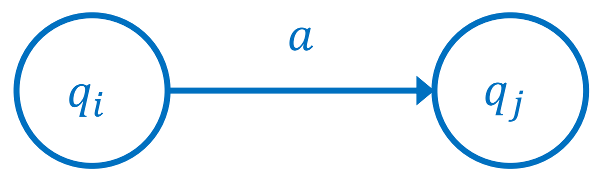
A start state



An accepting state



A state transition (e.g., $\delta(q_i, a) = q_j$)



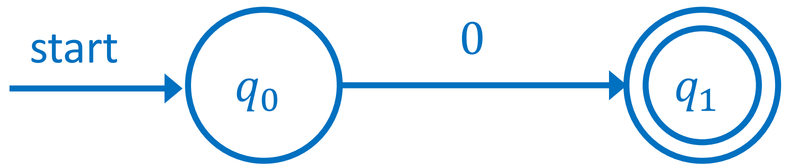
Finite automata

Simple examples

If $\Sigma = \{0\}$

- For a regular expression 0, where $L(0) = \{0\}$

  $$\boldsymbol{M = \{Q = \{q_0, q_1\}, \Sigma = \{0\}, \delta = \{\delta(q_0, 0) = q_1\}, q_0, F = \{q_1\}\}}$$



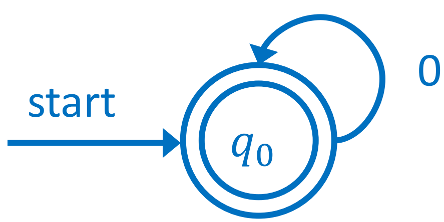
Finite automata

Simple examples

If $\Sigma = \{0\}$

- For a regular expression 0^*, where $L(0^*) = \{\epsilon, 0, 00, 000, \ldots\}$

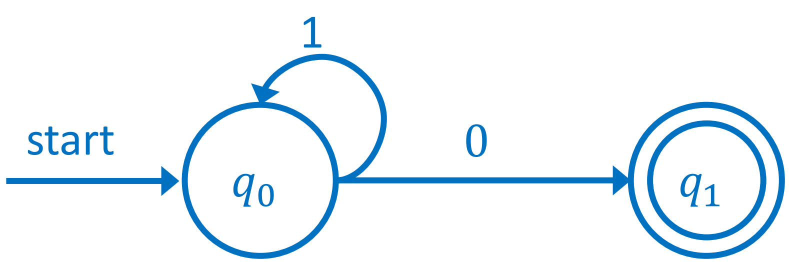
  $$\boldsymbol{M = \{Q = \{q_0\}, \Sigma = \{0\}, \delta = \{\delta(q_0, 0) = q_0\}, q_0, F = \{q_0\}\}}$$



Finite automata

Simple examples

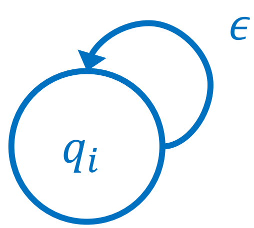
If $\Sigma = \{0, 1\}$, what is the regular expression this transition graph describes?



Finite automata

A special kind of state transition: ϵ-move

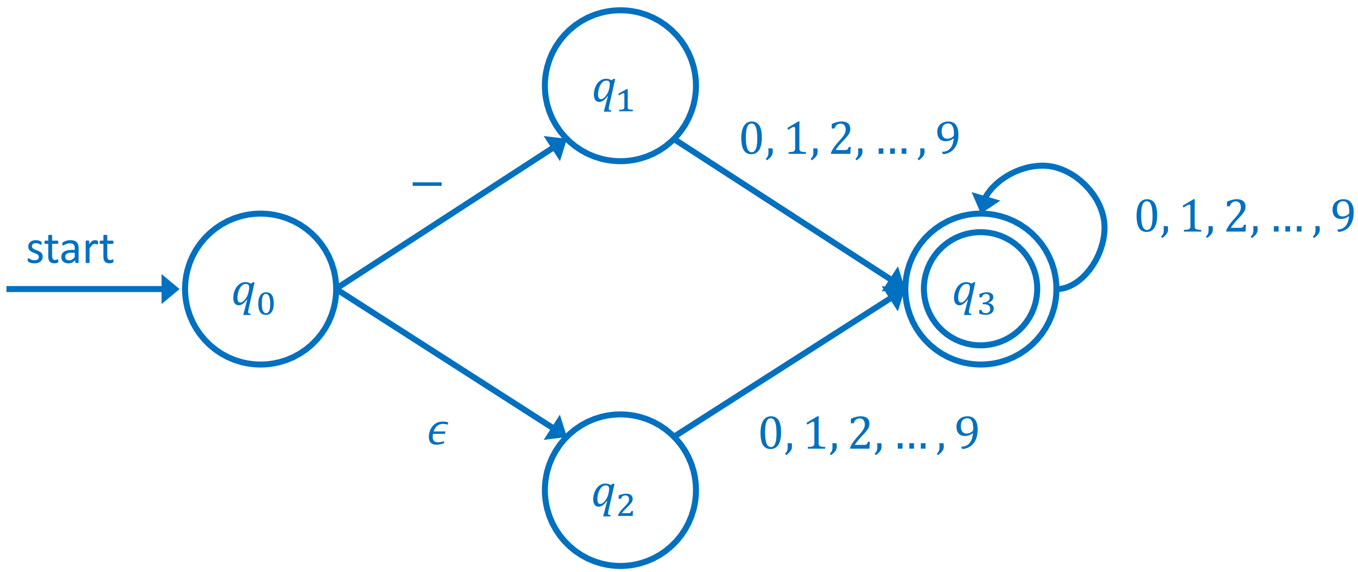
- A finite automata machine can move from q_i to q_j without reading inputs



Finite automata

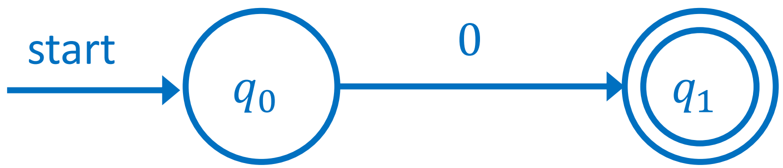
Simple examples

If $\Sigma = \{0, 1, 2, \ldots, 9, -\}$, what is the regular expression this transition describes?

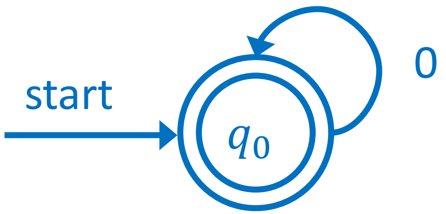


Finite automata

A finite automata can be also expressed in the form of table, a transition table



	0
$\boldsymbol{q_0}$	q_1
$\boldsymbol{q_1}$	$\emptyset$



	0
$\boldsymbol{q_0}$	q_0

	$\boldsymbol{\epsilon}$
$\boldsymbol{q_i}$	q_j
$\boldsymbol{q_j}$	$\emptyset$

Deterministic VS non-deterministic

Deterministic finite automata (DFA)

- (Exactly or at most) one transition for each state and for each input symbol
- No ϵ-moves

Non-deterministic finite automata (NFA)

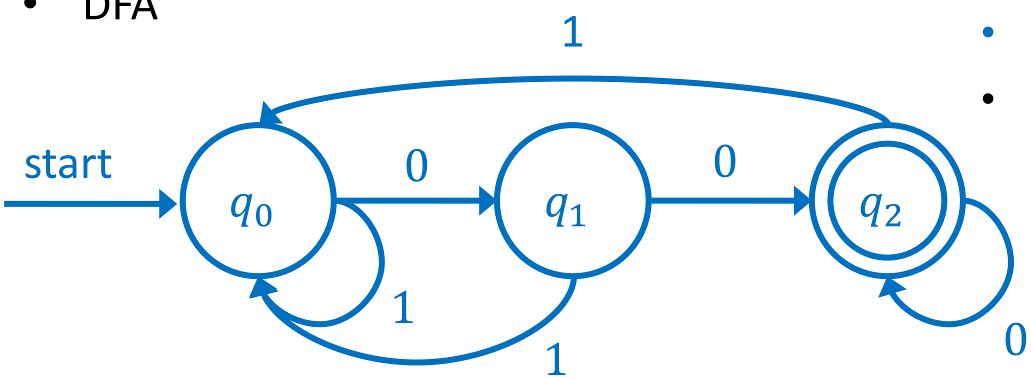
- Multiple transitions for each state and for each input symbol are allowed
- ϵ-moves are allowed

Deterministic VS non-deterministic

DFAs and NFAs can recognize the same set of regular languages

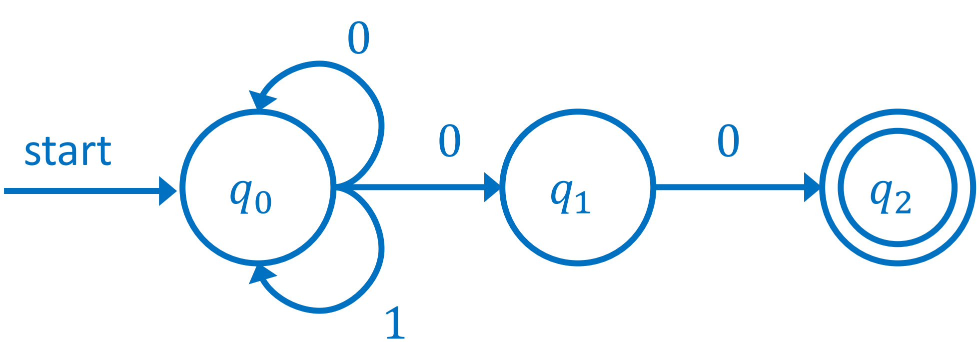
e.g., If $\Sigma = \{0,1\}$, for a regular expression $(0|1)^*00$

- DFA



- One deterministic path for a single input
- Accepted if and only if the path is from the start state to one of the final states

- NFA



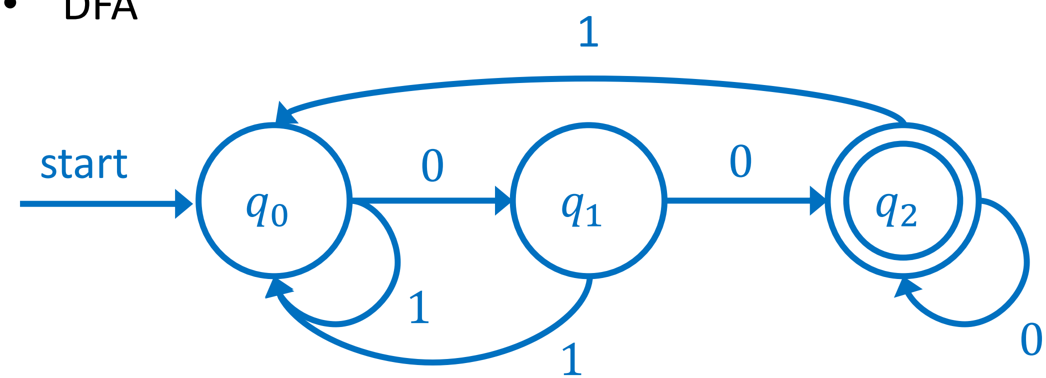
- Multiple possible paths for a single input
- Accepted if and only if any path among the possible paths is from the start state to one of the final states

Deterministic VS non-deterministic

DFAs and NFAs can recognize the same set of regular languages

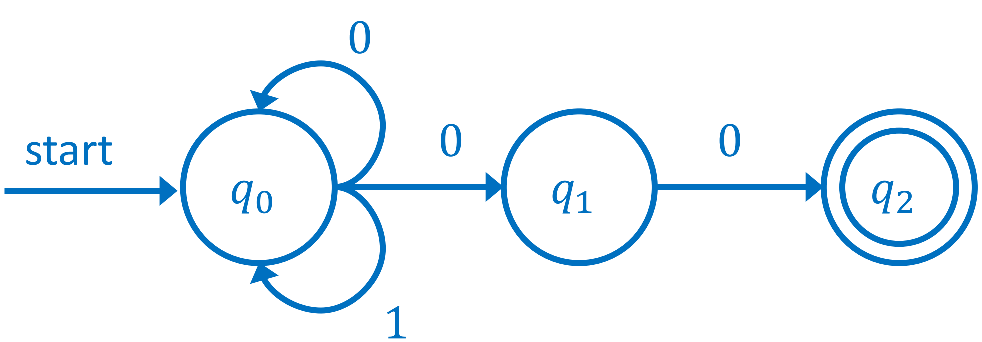
e.g., If $\Sigma = \{0,1\}$, for a regular expression $(0|1)^*00$

- DFA



Faster to execute!

- NFA



Simpler to represent!

Deterministic VS non-deterministic

	DFA	NFA
# of transitions per input per state	Zero or one	Zero or more
ϵ-move	X	O
# of path for a given input	Only one	One or more
Accepting condition	For a given input, its path must end in one of accepting states	For a given input, there must be at least one path ending in one of accepting states
Pros	Fast to execute (only one path)	Simple to represent (easy to make/understand)
Cons	Complex -> space problem (exponentially larger than NFA)	Slow -> performance problem (several paths)

Procedures for implementing lexical analyzers

Lexical specifications

Regular expressions

NFA

DFA (in the form of a transition table)

Regular expressions to NFAs

McNaughton-Yamada-Thompson algorithm (a.k.a., Thomson's construction)

This works recursively by splitting an expression into its constituent subexpressions

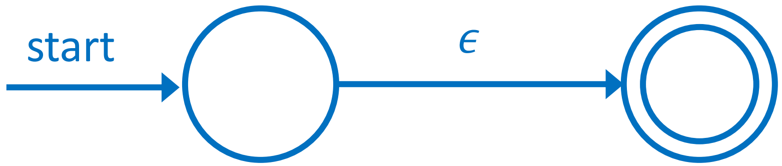
Examples

$$A|B$$

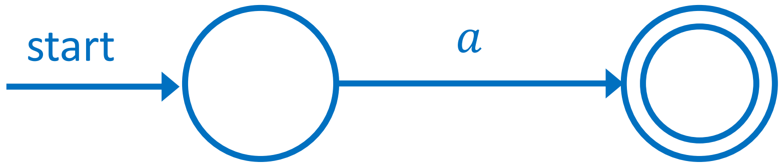
Regular expressions to NFAs

For a regular expression

- $\epsilon,\ L(\epsilon) = \{\epsilon\}$



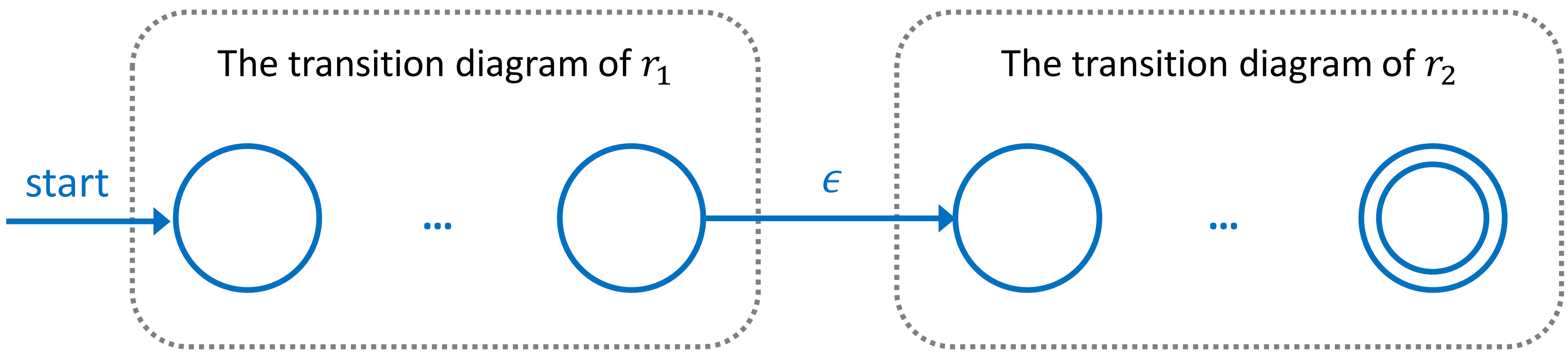
- $a,\ L(a) = \{a\}$



Regular expressions to NFAs

For a regular expression

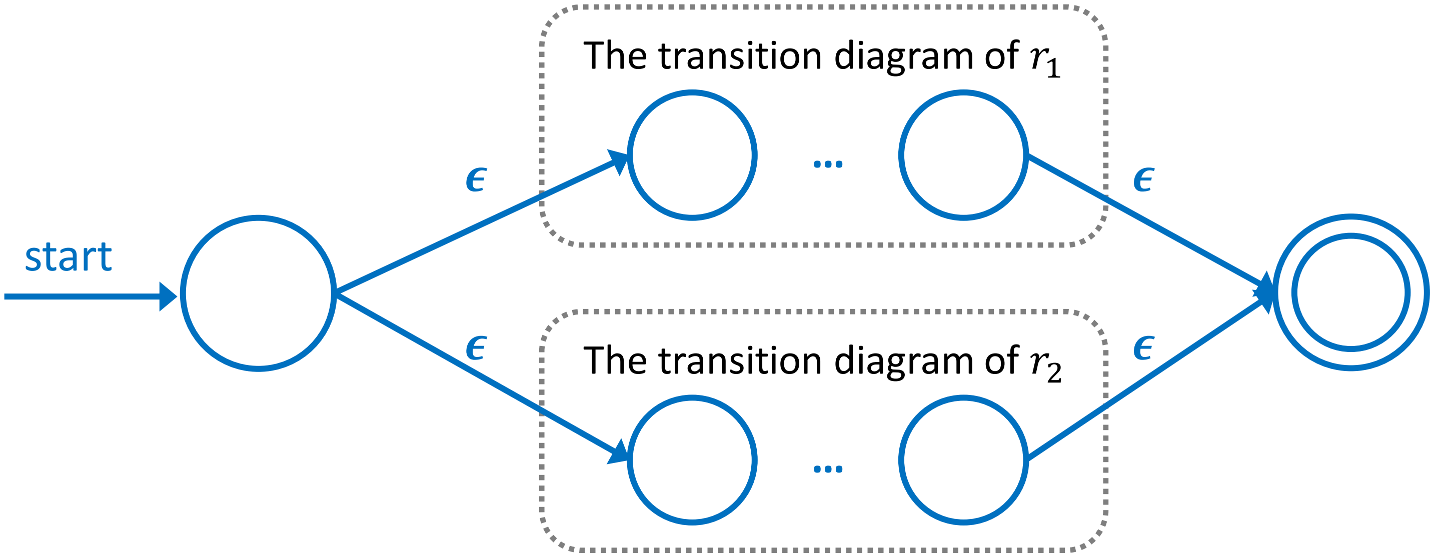
- $r_1 r_2$



Regular expressions to NFAs

For a regular expression

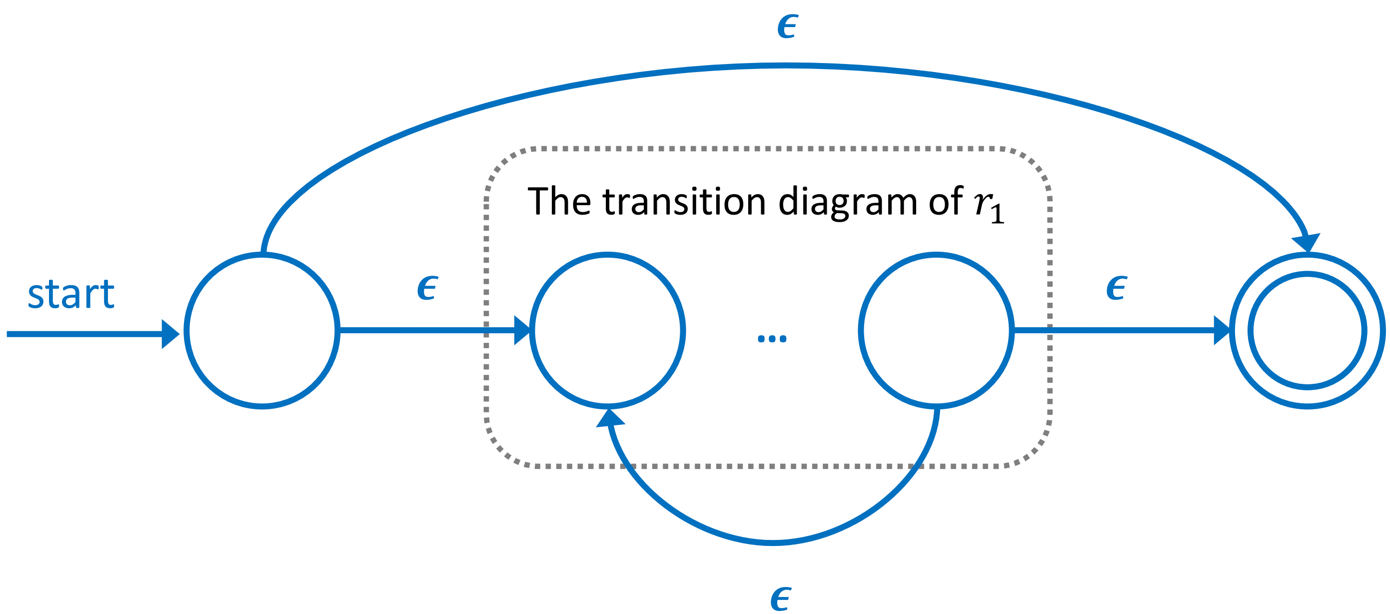
- $r_1 | r_2$



Regular expressions to NFAs

For a regular expression

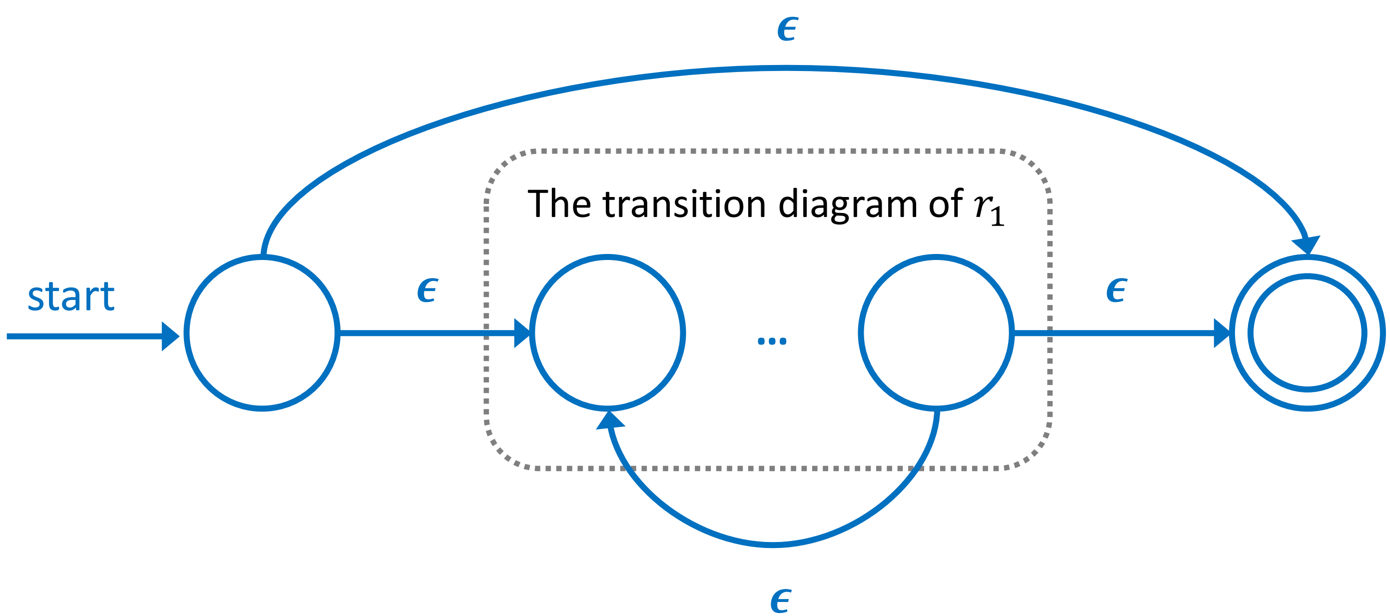
- r_1^*



Regular expressions to NFAs

For a regular expression

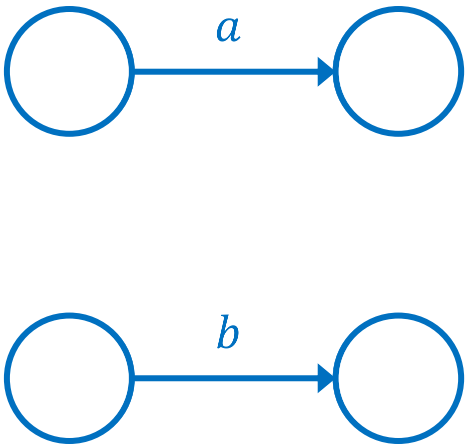
- Q. $r_1^+ = r_1 r_1^*$??



Regular expressions to NFAs

For a regular expression

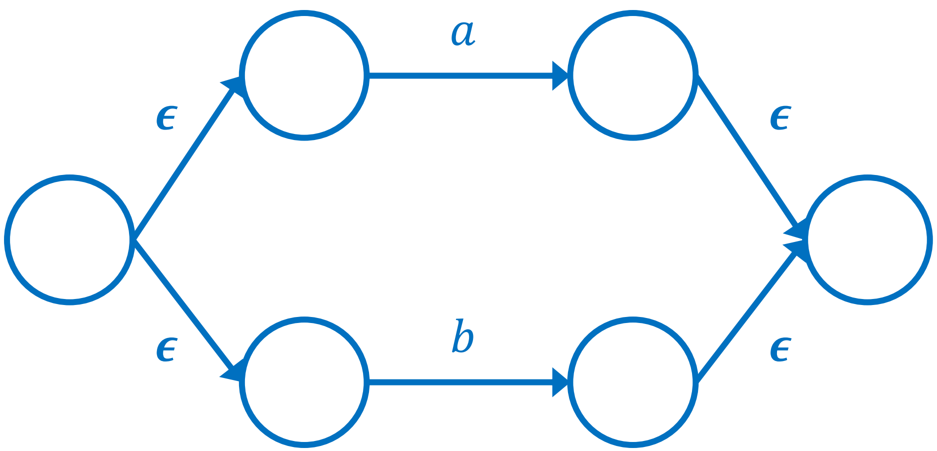
- e.g., $(a|b)^*c$



Regular expressions to NFAs

For a regular expression

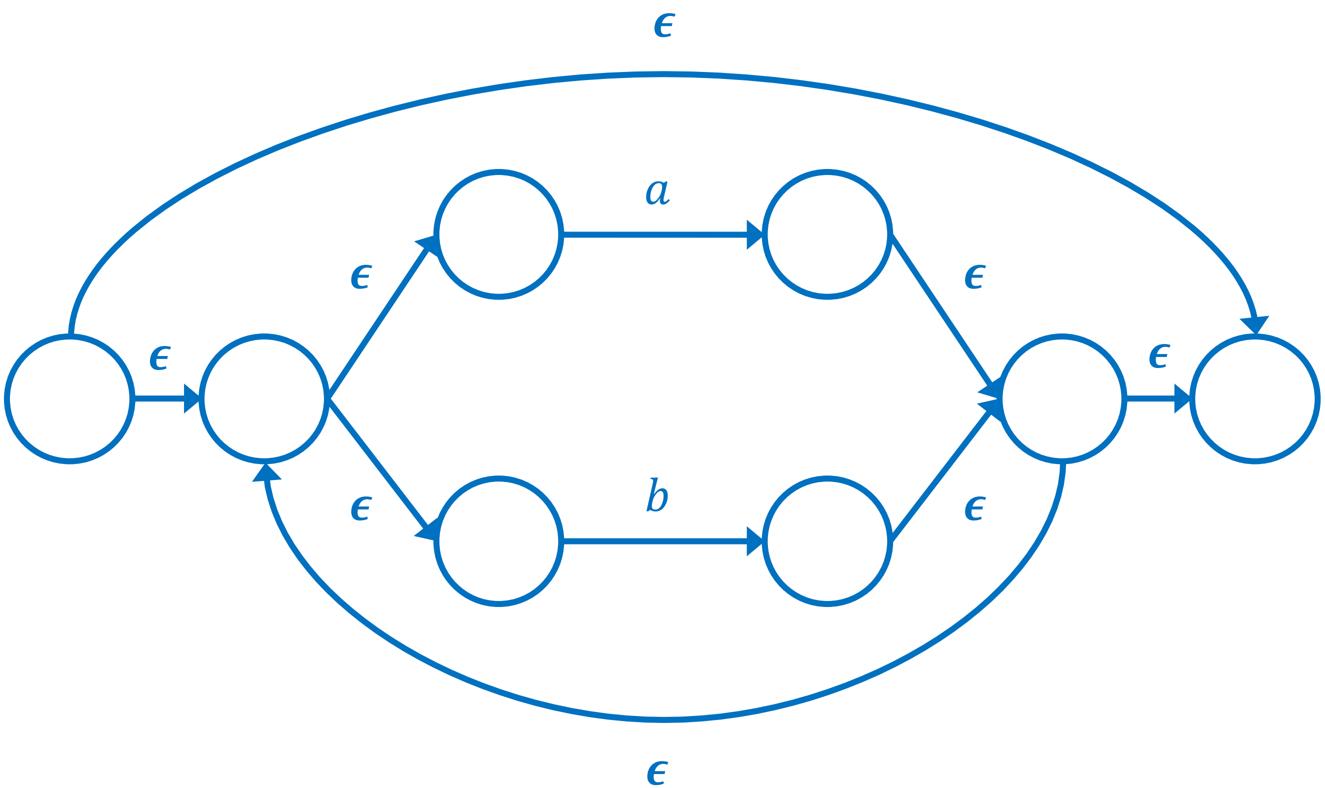
- e.g., $(a|b)^*c$



Regular expressions to NFAs

For a regular expression

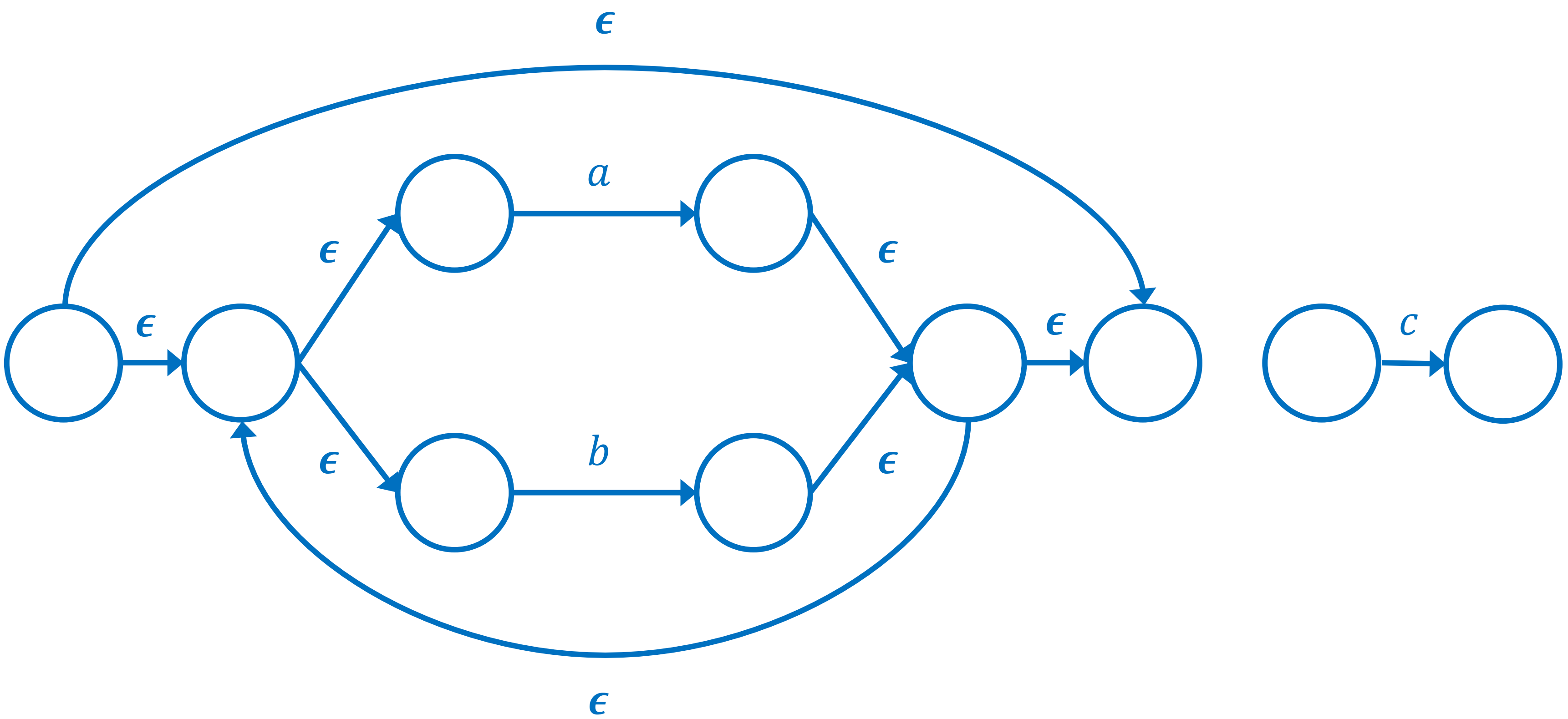
- e.g., $(a|b)^{*}c$



Regular expressions to NFAs

For a regular expression

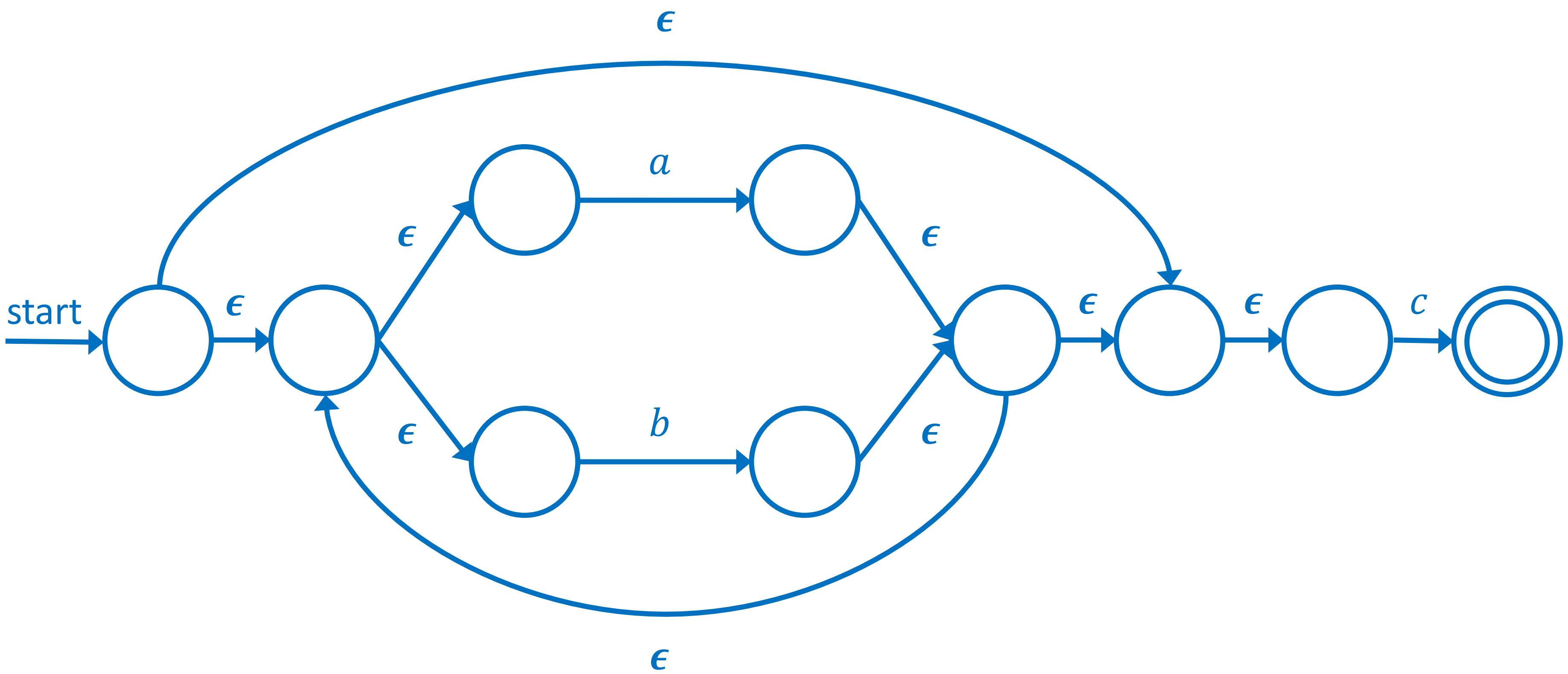
- e.g., $(a|b)^{*}c$



Regular expressions to NFAs

For a regular expression

- e.g., $(a|b)^*c$



Examples

$$L(sIdentifier) = \{a, \quad aA, \quad A, \quad Aa, \quad AC, \quad AC123, \quad A123a, \ldots\}$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

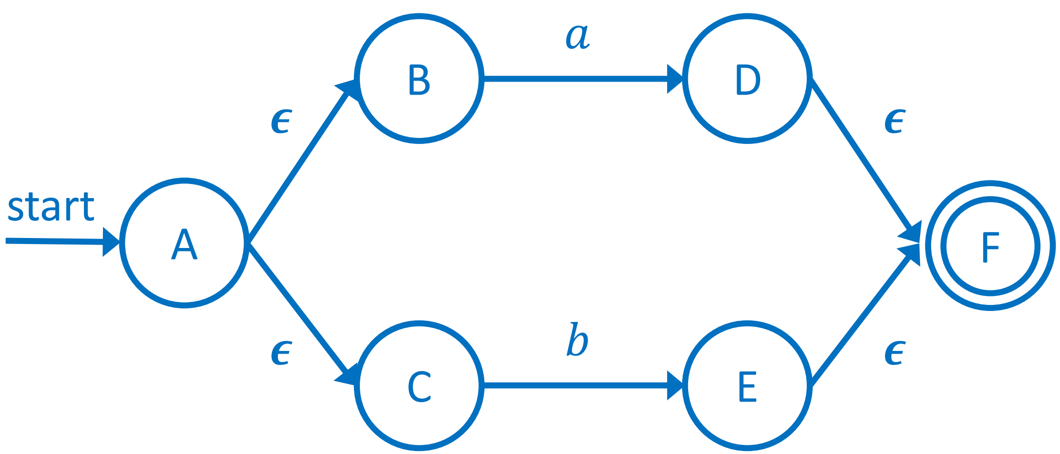
NFAs to DFAs

Subset (powerset) construction algorithm

- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Definitions

- $\boldsymbol{\epsilon\text{-}closure}(\boldsymbol{q^N})$**:** A set of NFA states reachable from NFA state q^N with only ϵ-moves
  ($\boldsymbol{q^N}$ is also included)
- $\boldsymbol{\epsilon\text{-}closure}(\boldsymbol{T})$: A set of NFA states reachable from some NFA state in a set $T = \{q_i, \ldots\}$ with only ϵ-moves



Examples

- $\epsilon\text{-}closure(A) = \{A, B, C\}$
- $\epsilon\text{-}closure(\{A, B, C, D\})$
  $= \{A, B, C, D, F\}$

NFAs to DFAs

Subset (powerset) construction algorithm

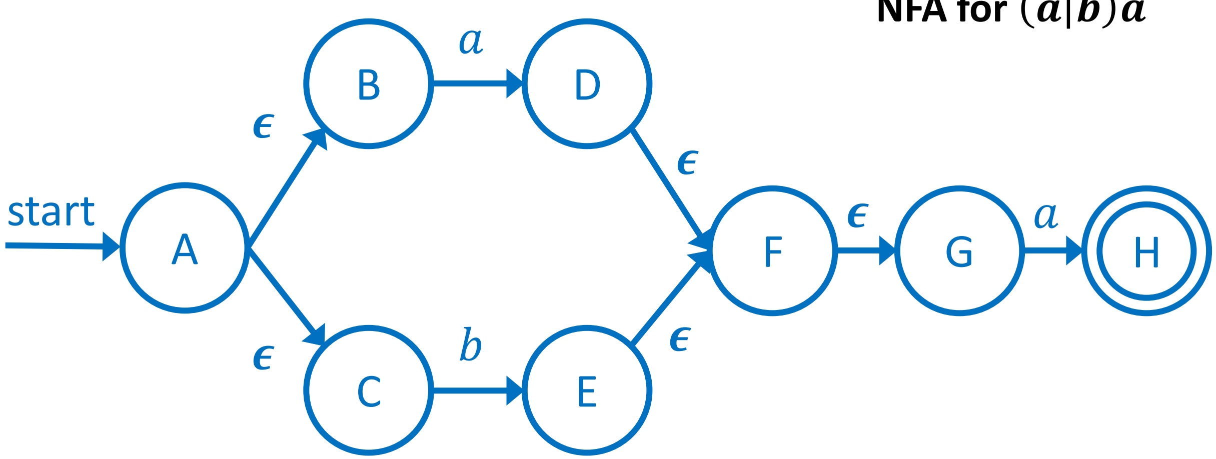
- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Step 1: Compute $\boldsymbol{\epsilon\text{-}closure(q_0^N)}$, where q_0^N is the start state of NFA

(Let denote the computation result as T_0)

$T_0 = \epsilon\text{-}closure(A) = \{A, B, C\}$

NFA for $(\boldsymbol{a}|\boldsymbol{b})\boldsymbol{a}$



NFAs to DFAs

Subset (powerset) construction algorithm

- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Step 2: Compute $\boldsymbol{\epsilon\text{-}closure(\delta(T_0, s))}$ for each input symbol s in Σ

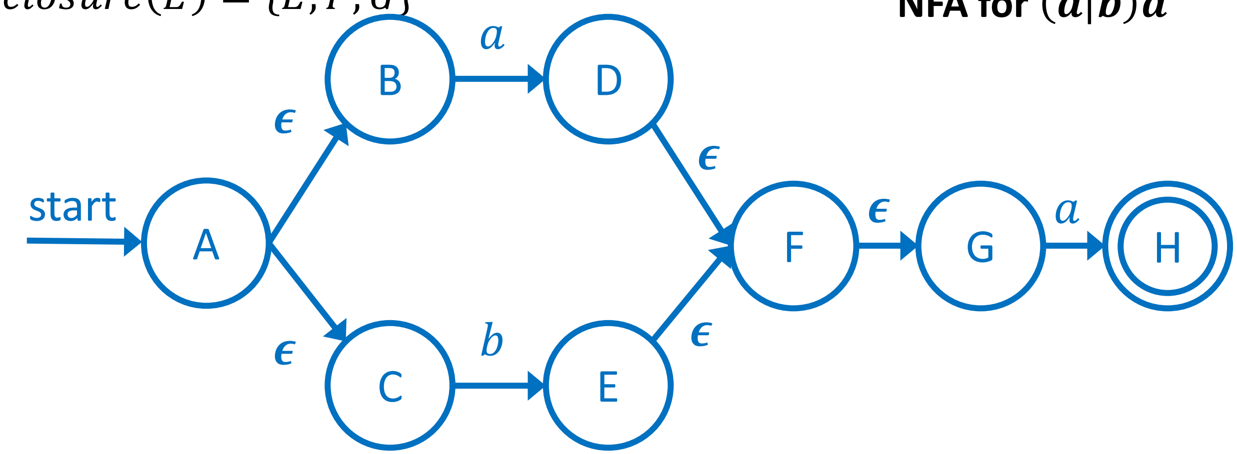
and denote the result as T_i iff $T_i \neq T_j$ $(0 \leq j < i)$

$$T_0 = \epsilon\text{-}closure(A) = \{A, B, C\}$$

$$T_1 = \epsilon\text{-}closure(\delta(T_0, a)) = \epsilon\text{-}closure(D) = \{D, F, G\}$$

$$T_2 = \epsilon\text{-}closure(\delta(T_0, b)) = \epsilon\text{-}closure(E) = \{E, F, G\}$$

NFA for $(\boldsymbol{a}|\boldsymbol{b})\boldsymbol{a}$



NFAs to DFAs

Subset (powerset) construction algorithm

- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Step 3: Repeat the step 2 for each new set of NFA states T_i until there is no more new result

$T_0 = \epsilon\text{-}closure(A) = \{A, B, C\}$

$T_1 = \epsilon\text{-}closure(\delta(T_0, a)) = \epsilon\text{-}closure(D) = \{D, F, G\}$

$T_2 = \epsilon\text{-}closure(\delta(T_0, b)) = \epsilon\text{-}closure(E) = \{E, F, G\}$

$T_3 = \epsilon\text{-}closure(\delta(T_1, a)) = \epsilon\text{-}closure(H) = \{H\}$

$\epsilon\text{-}closure(\delta(T_1, b)) = \emptyset$

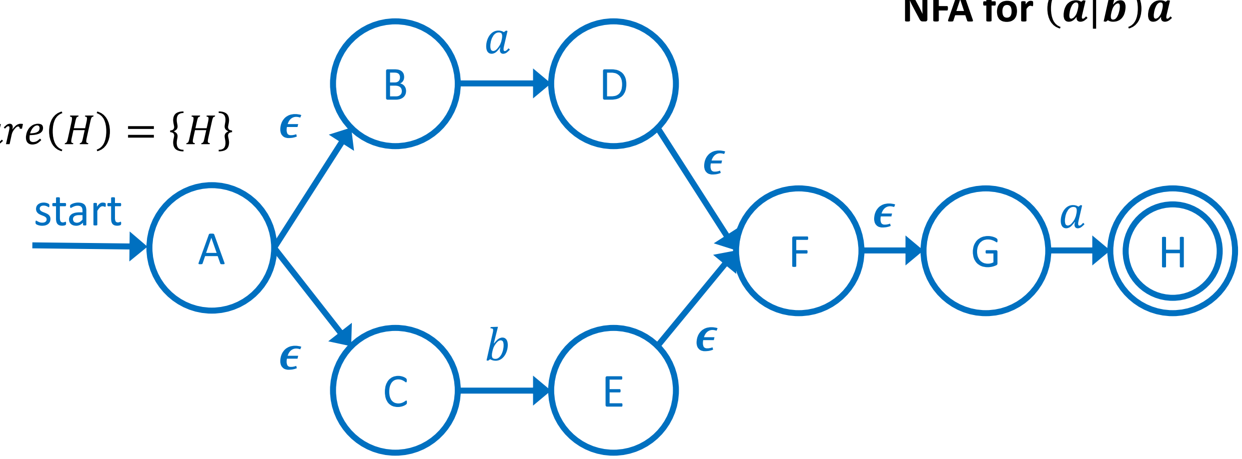
$\epsilon\text{-}closure(\delta(T_2, a)) = \epsilon\text{-}closure(H) = \{H\}$

$\epsilon\text{-}closure(\delta(T_2\ b)) = \emptyset$

$\epsilon\text{-}closure(\delta(T_3\ a)) = \emptyset$

$\epsilon\text{-}closure(\delta(T_3\ b)) = \emptyset$

NFA for $(a|b)a$



NFAs to DFAs

Subset (powerset) construction algorithm

- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Step 3: Repeat the step 2 for each new set of NFA states T_i until there is no more new result

$T_0 = \epsilon\text{-}closure(A) = \{A, B, C\}$

$T_1 = \epsilon\text{-}closure(\delta(T_0, a)) = \epsilon\text{–}closure(D) = \{D, F, G\}$

$T_2 = \epsilon\text{-}closure(\delta(T_0, b)) = \epsilon\text{–}closure(E) = \{E, F, G\}$

$T_3 = \epsilon\text{-}closure(\delta(T_1, a)) = \epsilon\text{–}closure(H) = \{H\}$

$\epsilon\text{-}closure(\delta(T_1, b)) = \emptyset$

$\epsilon\text{-}closure(\delta(T_2, a)) = \epsilon\text{–}closure(H) = \{H\}$

$\epsilon\text{-}closure(\delta(T_2\ b)) = \emptyset$

$\epsilon\text{-}closure(\delta(T_3\ a)) = \emptyset$

$\epsilon\text{-}closure(\delta(T_3\ b)) = \emptyset$

	$\boldsymbol{a}$	$\boldsymbol{b}$
$\boldsymbol{T_0}$	T_1	T_2
$\boldsymbol{T_1}$	T_3	$\emptyset$
$\boldsymbol{T_2}$	T_3	$\emptyset$
$\boldsymbol{T_3}$	$\emptyset$	$\emptyset$

NFAs to DFAs

Subset (powerset) construction algorithm

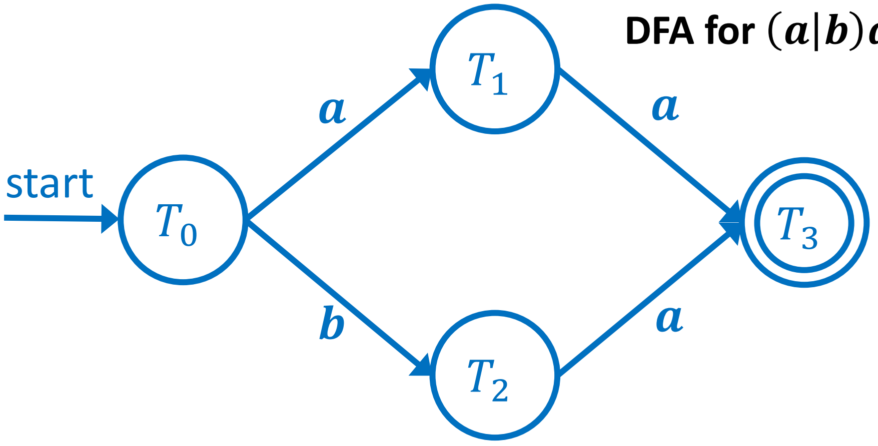
- **Basic idea:** Grouping a set of NFA states reachable after seeing some input strings

Step 4: Based on the computation results T_i, construct DFA as follows

- Each T_i is a DFA state
- T_0 is the start state of DFA
- Every T_i which includes any final state in NFA is the final state of DFA

	$\boldsymbol{a}$	$\boldsymbol{b}$
$\boldsymbol{T_0}$	T_1	T_2
$\boldsymbol{T_1}$	T_3	$\emptyset$
$\boldsymbol{T_2}$	T_3	$\emptyset$
$\boldsymbol{T_3}$	$\emptyset$	$\emptyset$

DFA for $(\boldsymbol{a}|\boldsymbol{b})\boldsymbol{a}$



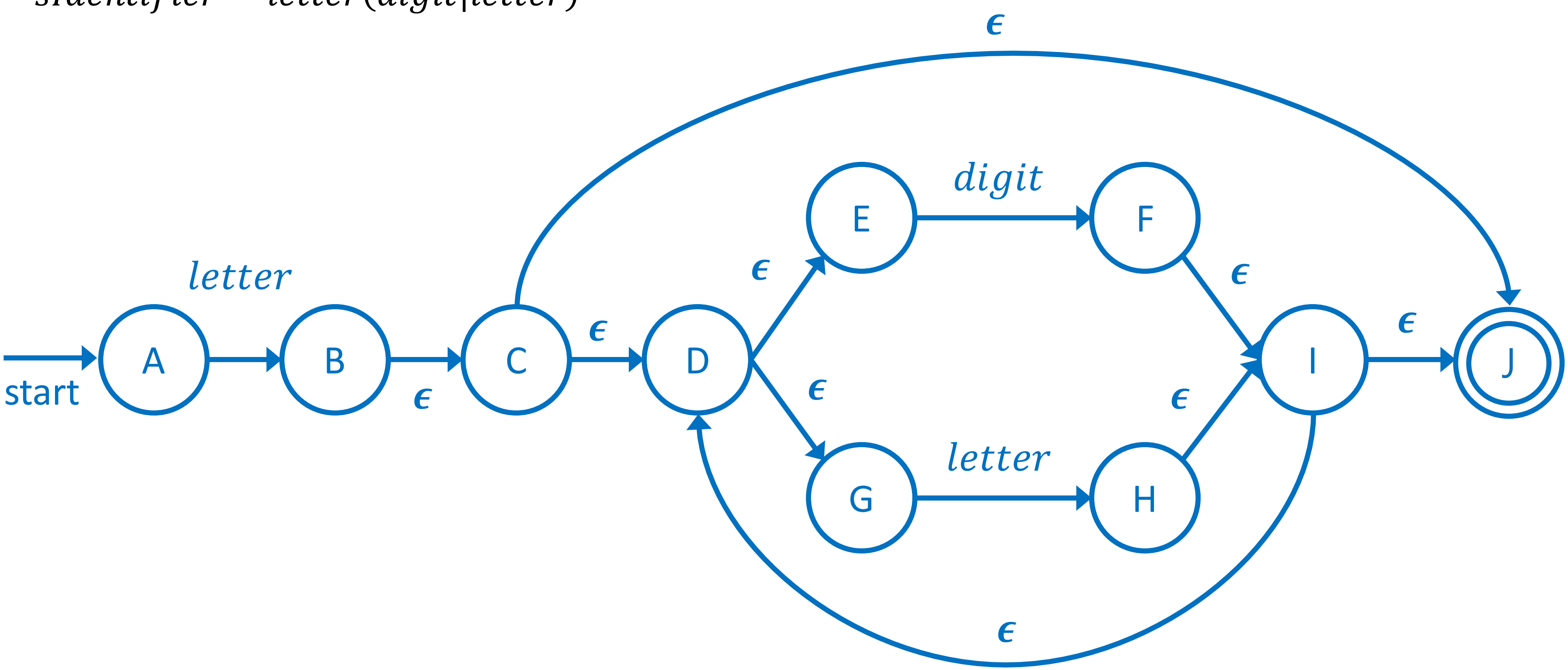
Examples

$$L(sIdentifier) = \{a, \quad aA, \quad A, \quad Aa, \quad AC, \quad AC123, \quad A123a, \dots\}$$

$letter = a|b|c|\dots|z|A|B|C|\dots|Z$

$digit = 0|1|2|\dots|9$

$sIdentifier = letter(digit|letter)^*$



Examples

$$L(sIdentifier) = \{a, \quad aA, \quad A, \quad Aa, \quad AC, \quad AC123, \quad A123a, \ldots\}$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$T_0 = \epsilon - closure(A) = \{A\}$

$T_1 = \epsilon - closure\big(\delta(T_0, letter)\big) = \{B, C, D, E, G, J\},\ \epsilon - closure\big(\delta(T_0, digit)\big) = \emptyset$

$T_2 = \epsilon - closure\big(\delta(T_1, letter)\big) = \{D, E, G, H, I, J\}$

$T_3 = \epsilon - closure\big(\delta(T_1, digit)\big) = \{D, E, G, F, I, J\}$

$\epsilon - closure\big(\delta(T_2, letter)\big) = \{D, E, G, H, I, J\} = T_2$

$\epsilon - closure\big(\delta(T_2, digit)\big) = \{D, E, G, F, I, J\} = T_3$

$\epsilon - closure\big(\delta(T_3, letter)\big) = \{D, E, G, H, I, J\} = T_2$

$\epsilon - closure\big(\delta(T_3, digit)\big) = \{D, E, G, H, I, J\} = T_3$

	$\boldsymbol{letter}$	$\boldsymbol{digit}$
$\boldsymbol{T_0}$	T_1	$\emptyset$
$\boldsymbol{T_1}$	T_2	T_3
$\boldsymbol{T_2}$	T_2	T_3
$\boldsymbol{T_3}$	T_2	T_3

Implementation of token recognizer

Lexical specifications

Regular expressions

NFA

DFA
(in the form of a transition table)

$mIdx = 0;$

$for\ 1 \leq i \leq n$

$\quad if\ a_1 a_2 \dots a_i \in L(Merged), mIdx = i$

end

$partition\ and\ classify\ a_1 a_2 \dots a_{mIdx}$

How can we do this task efficiently ?

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	$\boldsymbol{letter}$	$\boldsymbol{digit}$	**(**
$\boldsymbol{T_0}$	T_1	$\emptyset$	T_4
$\boldsymbol{T_1}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_2}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_3}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_4}$	$\emptyset$	$\emptyset$	$\emptyset$

For an input string COMPARE(A

Step	State	Input symbol	Next state
1	T_0	C	T_1

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	$\boldsymbol{letter}$	$\boldsymbol{digit}$	**(**
$\boldsymbol{T_0}$	T_1	Ø	T_4
$\boldsymbol{T_1}$	T_2	T_3	Ø
$\boldsymbol{T_2}$	T_2	T_3	Ø
$\boldsymbol{T_3}$	T_2	T_3	Ø
$\boldsymbol{T_4}$	Ø	Ø	Ø

For an input string COMPARE(A

Step	State	Input symbol	Next state
1	T_0	C	T_1
2	T_1	O	T_2
3	T_2	M	T_2
4	T_2	P	T_2
5	T_2	A	T_2
6	T_2	R	T_2
7	T_2	E	T_2

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	letter	***digit***	**(**
$\boldsymbol{T_0}$	T_1	∅	T_4
$\boldsymbol{T_1}$	T_2	T_3	∅
$\boldsymbol{T_2}$	T_2	T_3	∅
$\boldsymbol{T_3}$	T_2	T_3	∅
$\boldsymbol{T_4}$	∅	∅	∅

For an input string COMPARE(A

Step	State	Input symbol	Next state
1	T_0	C	T_1
2	T_1	O	T_2
3	T_2	M	T_2
4	T_2	P	T_2
5	T_2	A	T_2
6	T_2	R	T_2
7	T_2	E	T_2
8	T_2	(	

No transition rule!! Error

Find the last input which reaches at an accepting state

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\dots|z|A|B|C|\dots|Z$

$digit = 0|1|2|\dots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	$\boldsymbol{letter}$	$\boldsymbol{digit}$	**(**
$\boldsymbol{T_0}$	T_1	$\emptyset$	T_4
$\boldsymbol{T_1}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_2}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_3}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_4}$	$\emptyset$	$\emptyset$	$\emptyset$

For an input string COMPARE(A

Step	State	Input symbol	Next state
1	T_0	C	T_1
2	T_1	O	T_2
3	T_2	M	T_2
4	T_2	P	T_2
5	T_2	A	T_2
6	T_2	R	T_2
7	T_2	E	T_2
8	T_2	(	

Partition "COMPARE" and classify it as $\boldsymbol{sIdentifier}$

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	$\boldsymbol{letter}$	$\boldsymbol{digit}$	**(**
$\boldsymbol{T_0}$	T_1	$\emptyset$	T_4
$\boldsymbol{T_1}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_2}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_3}$	T_2	T_3	$\emptyset$
$\boldsymbol{T_4}$	$\emptyset$	$\emptyset$	$\emptyset$

For the remaining string (A

Step	State	Input symbol	Next state
1	T_0	(	T_4
2	T_4	A	

Partition "(" and classify it as $\boldsymbol{lparen}$

Implementation of token recognizer

Examples

$$Merged = sIdentifier|lparen$$

$letter = a|b|c|\ldots|z|A|B|C|\ldots|Z$

$digit = 0|1|2|\ldots|9$

$sIdentifier = letter(digit|letter)^*$

$lparen = ($

	$letter$	$digit$	(
T_0	T_1	$\emptyset$	T_4
T_1	T_2	T_3	$\emptyset$
T_2	T_2	T_3	$\emptyset$
T_3	T_2	T_3	$\emptyset$
T_4	$\emptyset$	$\emptyset$	$\emptyset$

For the remaining string A

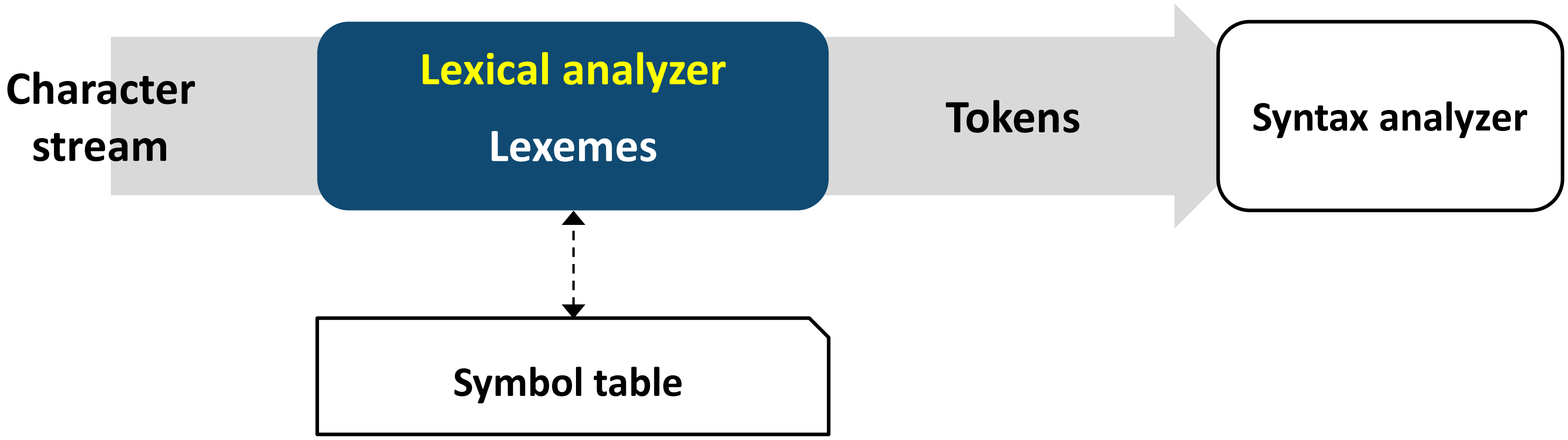
Step	State	Input symbol	Next state
1	T_0	A	T_1
2	T_1	End-of-input	

Partition "A" and classify it as $sIdentifier$

Summary

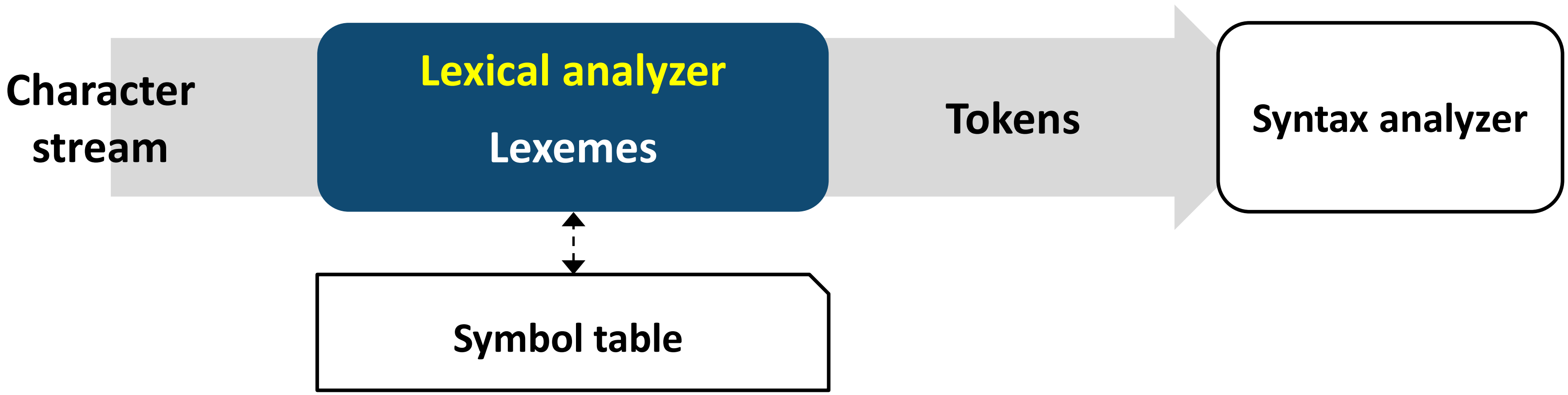
What does a lexical analyzer do?

1. Reading the input characters of a source program
2. Grouping the characters into meaningful sequences, called **lexemes**
3. Producing a sequence of **tokens**
4. Storing the token information into a symbol table
5. Sending the tokens to a syntax analyzer



Summary

What does a lexical analyzer do?



Remaining questions in designing lexical analyzers

1. How to specify the patterns for tokens? **Regular languages**

2. How to recognize the tokens from input streams? **Finite automata**