EC3204: Programming Languages and Compilers

Lecture 3 — Lexical Analysis (2)
Recognition of Tokens

Sunbeom So
Fall 2024

This Lecture: String Recognition using Finite Automata

- **Finite automata** are string recognizers that return either "yes" (accept) or "no" (reject).

- There are two types of finite automata.
  - **Non-deterministic Finite Automata (NFA)**
  - **Deterministic Finite Automata (DFA)**
- We will learn about:
  - Definitions of NFA and DFA.
  - How to recognize strings using NFA and DFA.

Appetizer: Possibility of Correct String Recognition

Given a string s and a lexical pattern R, to check whether $s \in L(R)$, we actually check whether $s \in L(FA)$ for some FA. Here, the recognition result is correct iff $L(R) = L(FA)$!

- If $L(R) \neq L(FA)$, we may accept (reject) invalid (valid) lexemes.

Appetizer: Possibility of Correct String Recognition

Given a string s and a lexical pattern R, to check whether $s \in L(R)$, we actually check whether $s \in L(FA)$ for some FA. Here, the recognition result is correct iff $L(R) = L(FA)$!

- If $L(R) \neq L(FA)$, we may accept (reject) invalid (valid) lexemes.

Before blindly using DFA or NFA, we would like to answer the question:

Q. Given R, does there exist FA such that $L(R) = L(FA)$?

In other words, is it possible to have a lexer that always outputs correct results? The answer is yes, as formalized by the theorem below.

Theorem (Kleene's Theorem)

Every language defined by a regex can be defined by NFA and DFA.[a]

[a] cf) The converse is true as well.

On top of this theoretical result, we can be confident that there is no gap between our goal and what we actually do!

Non-deterministic Finite Automata (NFA)

Definition (NFA)

A non-deterministic finite automaton (NFA) is defined as

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

where

- Q: a finite set of states
- Σ: an alphabet, i.e., a finite set of input symbols. We assume that $\epsilon \notin \Sigma$.
- $q_0 \in Q$: the initial state (start state)
- $F \subseteq Q$: a set of final states (accepting states)
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$: transition function that outputs a set of next states, for each state and for each symbol in $\Sigma \cup \{\epsilon\}$.

Example: NFA

- An example of NFA $M : (Q, \Sigma, \delta, q_0, F)$ according to the definition

$$(\{0, 1, 2, 3\}, \{a, b\}, \delta, 0, \{3\})$$

$$\begin{array}{llll}
\delta(0, a) = \{0, 1\} & \delta(1, a) = \emptyset & \delta(2, a) = \emptyset & \delta(3, a) = \emptyset \\
\delta(0, b) = \{0\} & \delta(1, b) = \{2\} & \delta(2, b) = \{3\} & \delta(3, b) = \emptyset
\end{array}$$

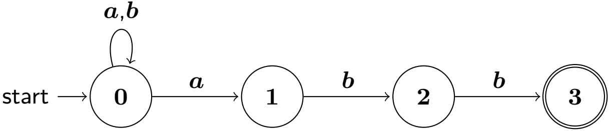
Example: NFA

- An example of NFA $M : (Q, \Sigma, \delta, q_0, F)$ according to the definition

$$(\{0,1,2,3\}, \{a,b\}, \delta, 0, \{3\})$$

$$\begin{array}{llll} \delta(0,a) = \{0,1\} & \delta(1,a) = \emptyset & \delta(2,a) = \emptyset & \delta(3,a) = \emptyset \\ \delta(0,b) = \{0\} & \delta(1,b) = \{2\} & \delta(2,b) = \{3\} & \delta(3,b) = \emptyset \end{array}$$

- A common representation method of NFA is **transition graph**.



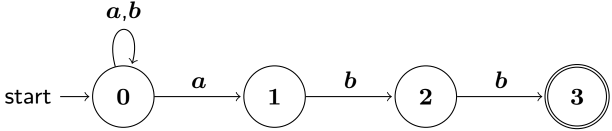
Example: NFA

- An example of NFA $M : (Q, \Sigma, \delta, q_0, F)$ according to the definition

$$(\{0,1,2,3\}, \{a,b\}, \delta, 0, \{3\})$$

$$\begin{array}{llll} \delta(0,a) = \{0,1\} & \delta(1,a) = \emptyset & \delta(2,a) = \emptyset & \delta(3,a) = \emptyset \\ \delta(0,b) = \{0\} & \delta(1,b) = \{2\} & \delta(2,b) = \{3\} & \delta(3,b) = \emptyset \end{array}$$

- A common representation method of NFA is **transition graph**.

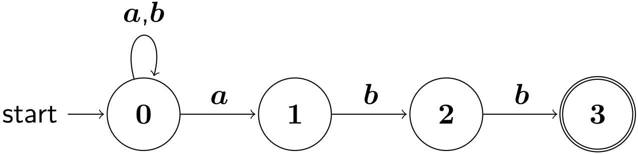


- NFA can also be represented by a transition table.

State	a	b	ϵ
0 (start)	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3 (final)	$\emptyset$	$\emptyset$	$\emptyset$

Strings Recognizable by NFA

The language of an NFA, denoted $L(NFA)$, is the set of strings recognizable by the NFA.



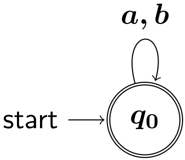
In our example,

$$L(NFA) = \{wabb \mid w \in \{a, b\}^*\} = L((a|b)^*abb).$$

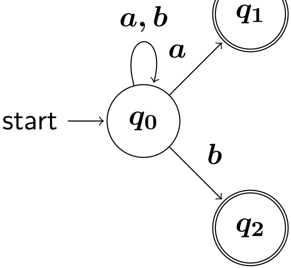
In words, the above NFA recognizes any strings that end with abb.

Find the languages (regular expressions) of the NFAs.

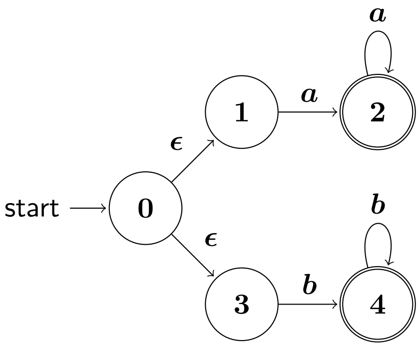
- $(a \mid b)^*$

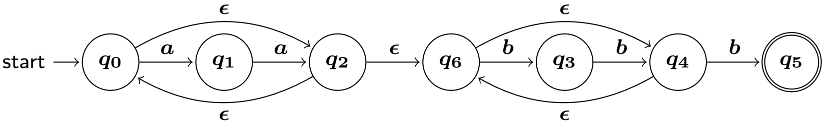


- $(a|b)^* \cdot (a|b)$, or $(a|b)^+$



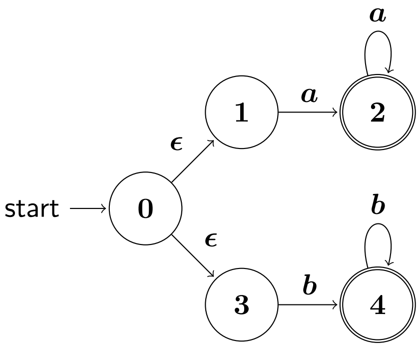
Exercise: Strings Recognizable by NFA

- 

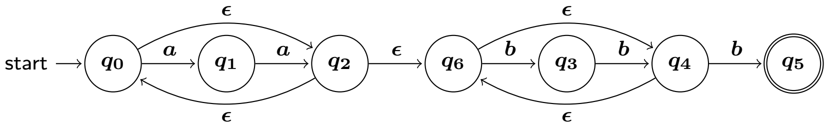
- 

Exercise: Strings Recognizable by NFA

- $(a^+|b^+)$

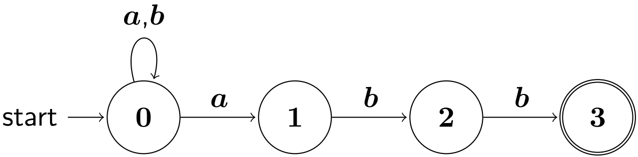


- $(aa)^*(bb)^*b$



String Recognition using NFA

An NFA recognizes a string w iff there is a path from the start state to one of the final states in the transition graph covered by w.



For example, a string $aabb$ is accepted ("yes") because we have a path ending at the final state 3:

$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

By contrast, a string aa is rejected ("no") because any path labeled aa does not end at the final state 3:

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0, \quad 0 \xrightarrow{a} 0 \xrightarrow{a} 1$$

Deterministic Finite Automata (DFA)

A DFA is a special case of an NFA, where

1. there are no moves on ϵ, and
2. for each state and input symbol, the next state is unique.

Definition (DFA)

A deterministic finite automaton (or DFA) is defined as

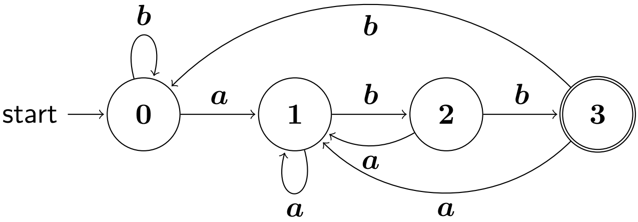
$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- Q: a finite set of states
- Σ: a finite set of input symbols (or input alphabet)
- $\delta : Q \times \Sigma \to Q$: a transition function that is a total function (defined for every possible input)
- $q_0 \in Q$: the initial state
- $F \subseteq Q$: a set of final states

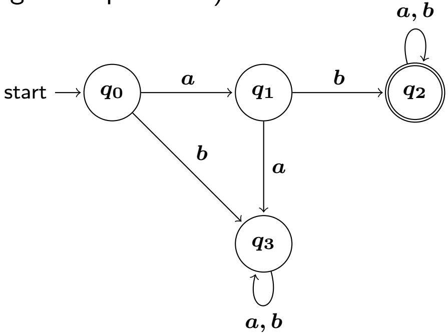
Example: DFA in Transition Graph

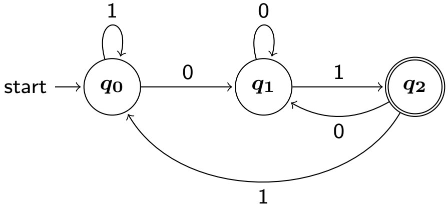
A DFA that accepts strings described by $(\boldsymbol{a} \mid \boldsymbol{b})^* \boldsymbol{abb}$.



Exercises

Find the languages (regular expressions) of the DFA.

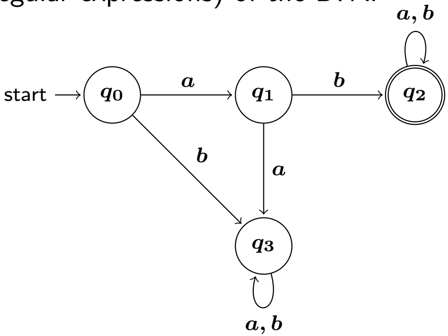
- 

- 

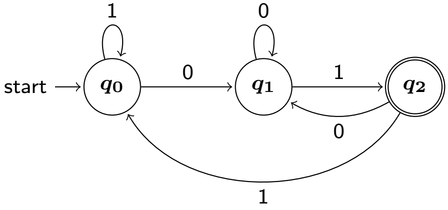
Exercises

Find the languages (regular expressions) of the DFA.

- $ab(a \mid b)^*$

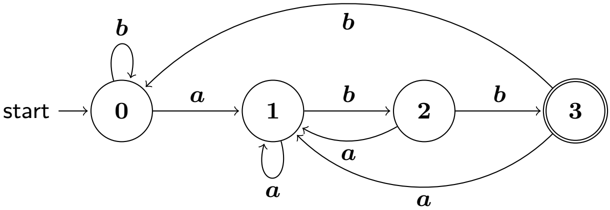


- $(0 \mid 1)^*01$



String Recognition using DFA

As in NFA, a DFA recognizes a string w iff there is a path from the start state to one of the final states in the transition graph covered by w.



For example, a string $aabb$ is accepted ("yes") because we have a path ending at the final state $\mathbf{3}$:

$$\mathbf{0} \xrightarrow{a} \mathbf{1} \xrightarrow{a} \mathbf{1} \xrightarrow{b} \mathbf{2} \xrightarrow{b} \mathbf{3}$$

By contrast, a string aa is rejected ("no") because the path labeled aa does not end at the final state $\mathbf{3}$:

$$\mathbf{0} \xrightarrow{a} \mathbf{1} \xrightarrow{a} \mathbf{1}$$

Why DFA?

Both NFA and DFA are string recognizers, but lexers often rely on DFA. Why?

Automaton	Initial (Construction)	Per String
NFA	$O(\|r\|)$	$O(\|r\| \times \|x\|)$
DFA typical case	$O(\|r\|^3)$	$O(\|x\|)$
DFA worst case	$O(\|r\|^2 2^{\|r\|})$	$O(\|x\|)$

$|r|$: the size of a regex r (#operators in r + #operands in r)

$|x|$: the size of an input string x

- One main reason: efficiency
- If a string recognizer to be built will be used many times repeatedly (e.g., lexical analysis), converting to a DFA is worthwhile.
- If a string recognizer to be built will be used only a few times (e.g., regex matching using `grep`), directly simulating an NFA would be more efficient.

Summary

We learned how to accept (resp., reject) the valid (resp., invalid) lexical patterns.

- Definitions of NFA and DFA.
- How to recognize strings using NFA and DFA.

Next class: how to construct string recognizers (NFA, DFA).