CS304: Automata and Formal Languages

Lec 9

Kleene's Theorem

Rachit Nimavat

August 21, 2025

Outline

Deterministic Finite Automata (DFA)

Formal Definition

DFA is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$

'symbol'	description
Q	finite set of states
Σ	underlying finite alphabet
δ	transition function between states
$q_0 \in Q$	start state
$F \subseteq Q$	accepting states

Deterministic Finite Automata (DFA)

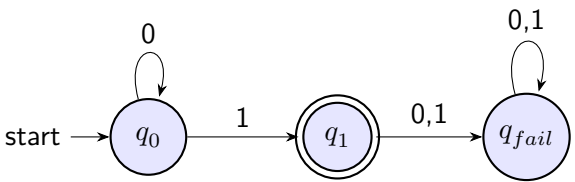
Visualizing DFA

State Diagram is the intuitive way we visualize and work with DFAs

Conventions:

- States Q are circles
- Start state (q_0) has an incoming arrow labeled start
- Accepting states (F) are double circles
- Transitions (δ) are arrows between states, labeled with input symbols from Σ

Our DFA for $L = 0^*1$:



Non-Deterministic Finite Automata (NFA)

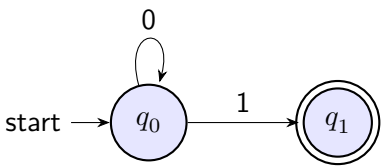
Visualizing NFA

State Diagram is the intuitive way we visualize and work with NFAs

Conventions:

- States Q are circles
- Start state (q_0) has an incoming arrow labeled start
- Accepting states (F) are double circles
- Transitions (δ) are arrows between states, labeled with input symbols from Σ

Our NFA for $L = 0^*1$:



NFA vs DFA

NFA

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

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$

Multiple Transitions. Possibly multiple outgoing arrows for same symbol

Zero Transitions. Possibly no outgoing arrow for a symbol (that path "dies" has license to kill)

ε-Transitions. Can change state without consuming an input symbol

DFA

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

$\delta : Q \times \Sigma \mapsto Q$

No such powers!

Theorem 1.1

A language is recognized by an NFA if and only if it is recognized by a DFA.

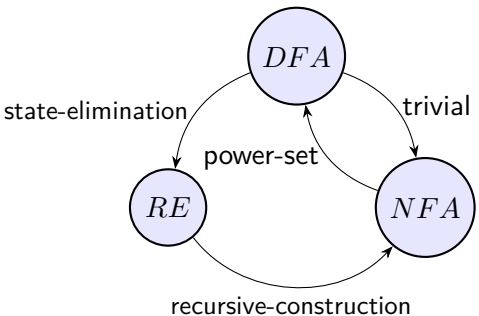
Regular Language

Defn: A language L is called a regular language if and only if there exists a regular expression R that describes it: $L = L(R)$

Theorem 2.1 (Kleene's Theorem)

All these definitions are equivalent:

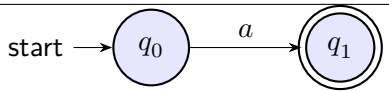
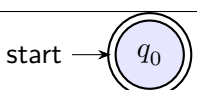
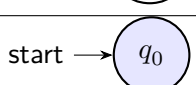
- $\exists$ *a DFA* D *such that* $L = L(D)$
- $\exists$ *an NFA* A *such that* $L = L(A)$
- $\exists$ *an RE* R *such that* $L = L(R)$



Kleene's Theorem: RE $\Longrightarrow$ NFA

Step-by-step algorithm to convert an RE to an NFA

Primitive Regular Expressions (Base case):

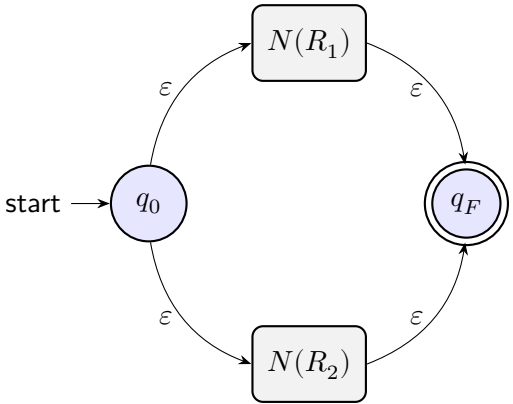
RE	NFA
a (where $a \in \Sigma$)	
ϵ	
$\emptyset$	

Kleene's Theorem: RE $\Longrightarrow$ NFA

Step-by-step algorithm to convert an RE to an NFA

Combining Machines (Inductive step):

- $R_1 + R_2$ (union)



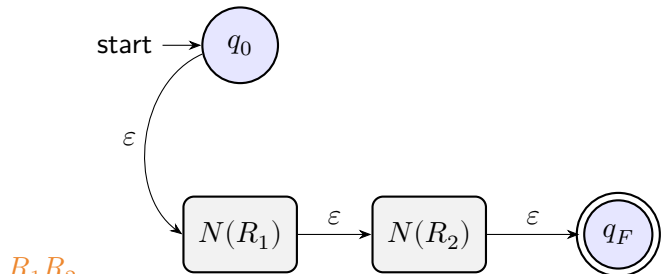
$R_1 + R_2$

Kleene's Theorem: RE $\Longrightarrow$ NFA

Step-by-step algorithm to convert an RE to an NFA

Combining Machines (Inductive step):

- $R_1 R_2$
(concatenation)



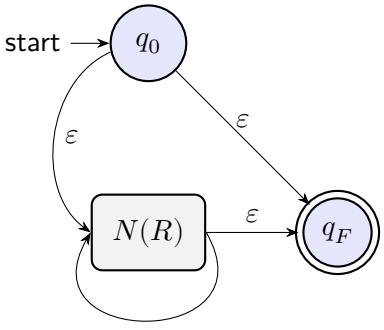
$R_1 R_2$

Kleene's Theorem: RE $\Longrightarrow$ NFA

Step-by-step algorithm to convert an RE to an NFA

Combining Machines (Inductive step):

- R_1^* (Kleene star)



R^*

NFA++

We have already shown that RE $\implies$ NFA.

Observation: An "NFA" where labels of its paths are REs is also an NFA!

Definition 4.1

Consider a finite alphabet Σ and let $\mathcal{R}$ be the set of all Regular Expressions over Σ. An NFA++ is a 5-tuple $A = (Q, \Sigma, \delta, q_0, q_f)$, where

Q: finite set of states

q_0: single starting state

q_f: single accepting state

$\delta : (Q \backslash q_f) \times (Q \backslash q_0) \mapsto \mathcal{R}$ recall, $\mathcal{R}$ includes the "empty" language $\emptyset$

HW: Formally prove that NFA = NFA++.

i.e. $\forall$NFA M_1, $\exists$NFA++ M_2 such that $L(M_1) = L(M_2)$.

Notice that the other direction: $\forall$NFA++ M_2, $\exists$NFA M_1 such that $L(M_2) = L(M_1)$ is trivial.

Kleene's Theorem: NFA++ $\implies$ RE

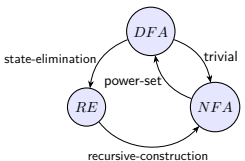
State Elimination

We'll remove states from NFA++ one-by-one until we are left with just the start state q_0 and accept states q_f

Over to the board!

What Now?

We completed one of the most important results of this course: Kleene's Theorem.



Closure Properties: A set is closed under an operation if applying that operation to its elements results in elements that are also in the set

- Integers are closed under addition int + int = int
- Integers are closed under division **false!** 3 / 2 = 1.5
- Real numbers are closed under division **false!** 0 / 0 is undefined
- Regular languages are closed under union They are regular expressions!
- Regular languages are closed under complementation They are DFAs!

HW: Prove last 2 statements!

Irregular Languages

We've spent all this time defining what regular languages are

But are there languages that are not regular?

Are there languages that a finite automaton cannot solve no matter how many states it has?

Yes! These are called **Irregular Languages**

How do we even prove that for some given language L, there is NO NFA that accepts it?

Pumping Lemma!