NFAs continued, Closure Properties of Regular Languages

Lecture 5

Tuesday, September 8, 2020

LaTeXed: July 23, 2020 13:41

5.1
Equivalence of NFAs and DFAs

Regular Languages, DFAs, NFAs

Theorem

Languages accepted by DFAs, NFAs, and regular expressions are the same.

- DFAs are special cases of NFAs (easy)
- NFAs accept regular expressions (seen)
- DFAs accept languages accepted by NFAs (shortly)
- Regular expressions for languages accepted by DFAs (later in the course)

Regular Languages, DFAs, NFAs

Theorem

Languages accepted by DFAs, NFAs, and regular expressions are the same.

- DFAs are special cases of NFAs (easy)
- NFAs accept regular expressions (seen)
- DFAs accept languages accepted by NFAs (shortly)
- Regular expressions for languages accepted by DFAs (later in the course)

Equivalence of NFAs and DFAs

Theorem

For every NFA N there is a DFA M such that $L(M) = L(N)$.

5.1.1
The idea of the conversion of NFA to DFA

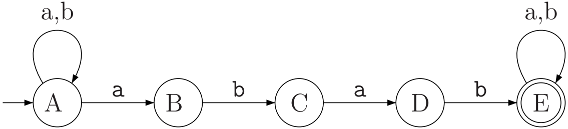
DFAs are memoryless...

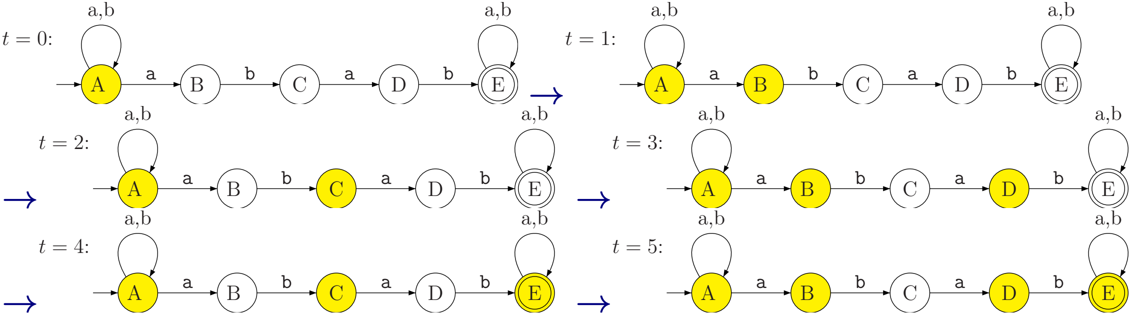
1. DFA knows only its current state.
2. The state is the memory.
3. To design a DFA, answer the question:
   What minimal info needed to solve problem.

Simulating NFA

Example the first revisited

Previous lecture.. Ran NFA[(N1)] on input ***ababa***.

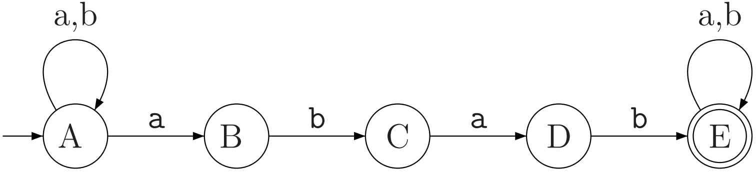




The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

(N1)



configuration: A set of states the automata might be in.

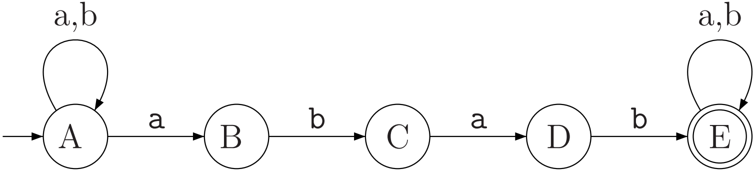
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

Big idea: Build a DFA on the configurations.

The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

(N1)



configuration: A set of states the automata might be in.

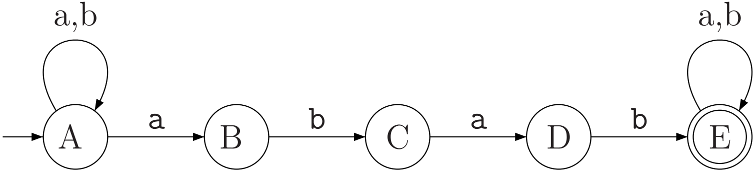
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

Big idea: Build a DFA on the configurations.

The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

(N1)



configuration: A set of states the automata might be in.

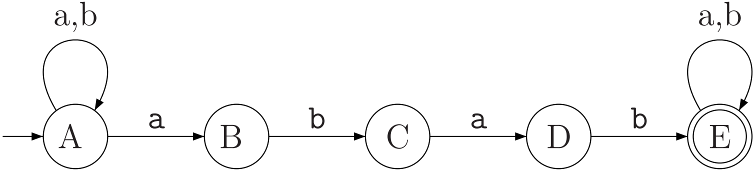
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

Big idea: Build a DFA on the configurations.

The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

(N1)

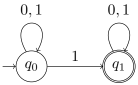


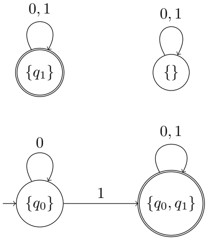
configuration: A set of states the automata might be in.

Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

Big idea: Build a DFA on the configurations.

Example





Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol a in the input.
- When should the program accept a string w? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: DFA M simulating N should know current configuration of N.

State space of the DFA is $\mathcal{P}(Q)$.

Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol a in the input.
- When should the program accept a string w? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: DFA M simulating N should know current configuration of N.

State space of the DFA is $\mathcal{P}(Q)$.

Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol a in the input.
- When should the program accept a string w? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: DFA M simulating N should know current configuration of N.

State space of the DFA is $\mathcal{P}(Q)$.

Simulating an NFA by a DFA

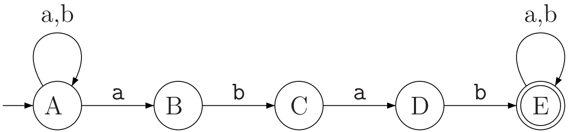
- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol a in the input.
- When should the program accept a string w? If $\delta^*(s, w) \cap A \neq \emptyset$.

Key Observation: DFA M simulating N should know current configuration of N.

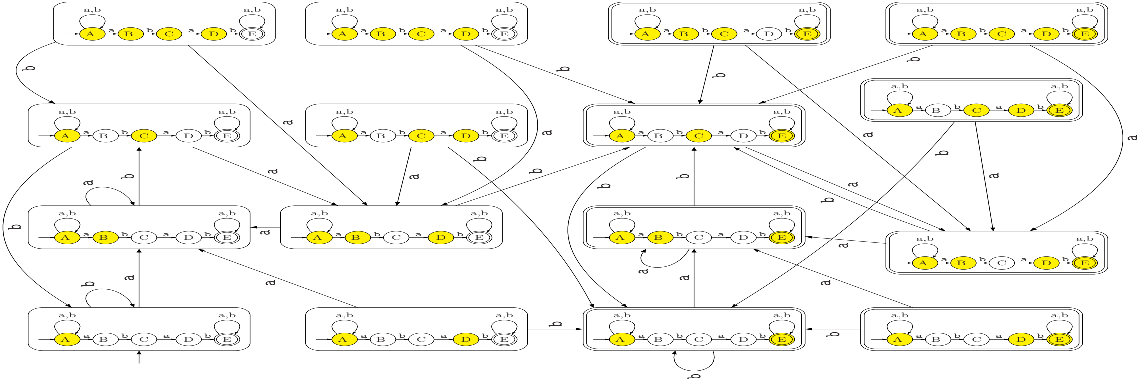
State space of the DFA is $\mathcal{P}(Q)$.

Example: DFA from NFA

NFA: (N1)



DFA:



Formal Tuple Notation for NFA

Definition

A non-deterministic finite automata (NFA) $N = (Q, \Sigma, \delta, s, A)$ is a five tuple where

- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q),
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

$\delta(q, a)$ for $a \in \Sigma \cup \{\epsilon\}$ is a subset of Q — a set of states.

THE END

...

(for now)