NFAs continued, Closure Properties of Regular Languages

Lecture 5

Tuesday, September 10, 2024

LATEXed: October 8, 2024 20:53

5.1
Equivalence of NFAs and DFAs

Regular Languages, DFAs, NFAs

Theorem 5.1.
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 5.2.

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

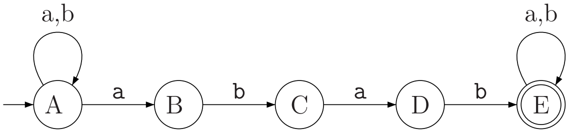
DFAs are memoryless...

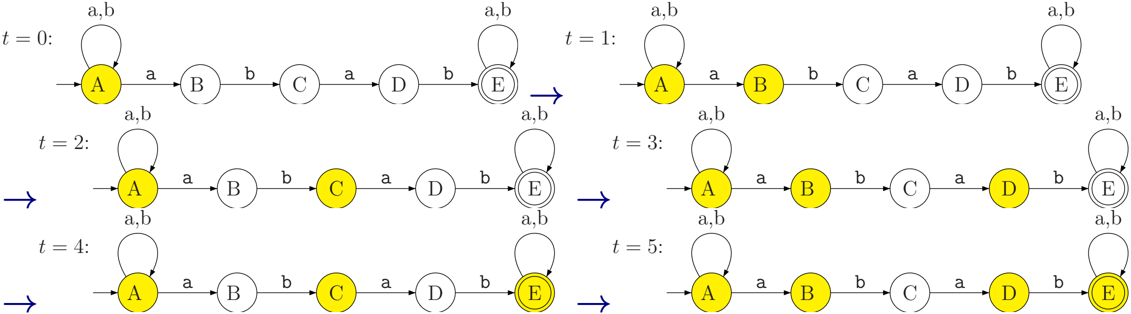
1. DFA knows only its current state.
2. The state is the memory.
3. To design a DFA, answer the question:
   What minimal info needed to solve problem.

Simulating NFA

Example the first revisited

Previous lecture.. Ran NFA $^{(N1)}$ on input ***ababa***.

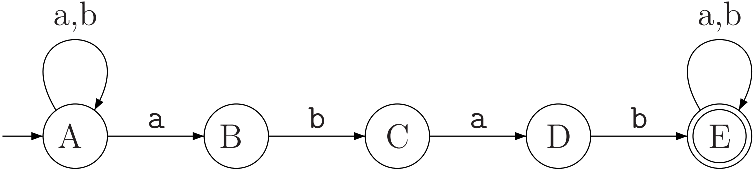




The state of the NFA

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

(N1)

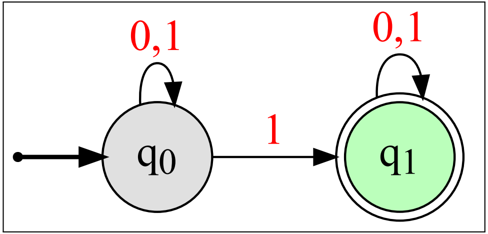


configuration: A set of states the automata might be in.
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...
Big idea: Build a DFA on the configurations.

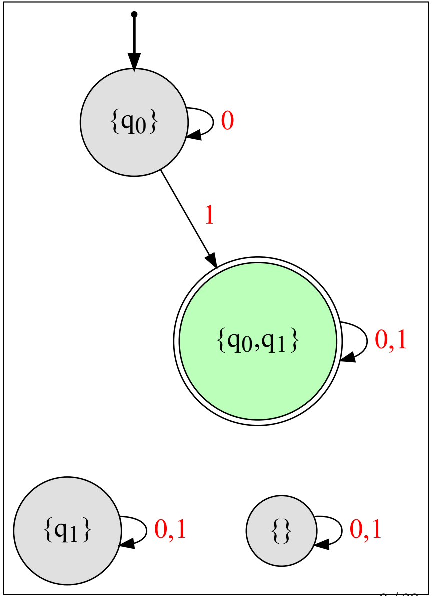
Example: Subset construction

NFA:

DFA:



$\Longrightarrow$



Simulating an NFA by a DFA

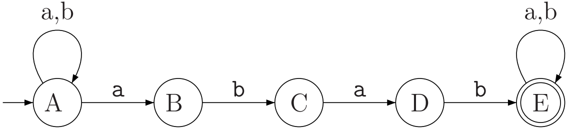
- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after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

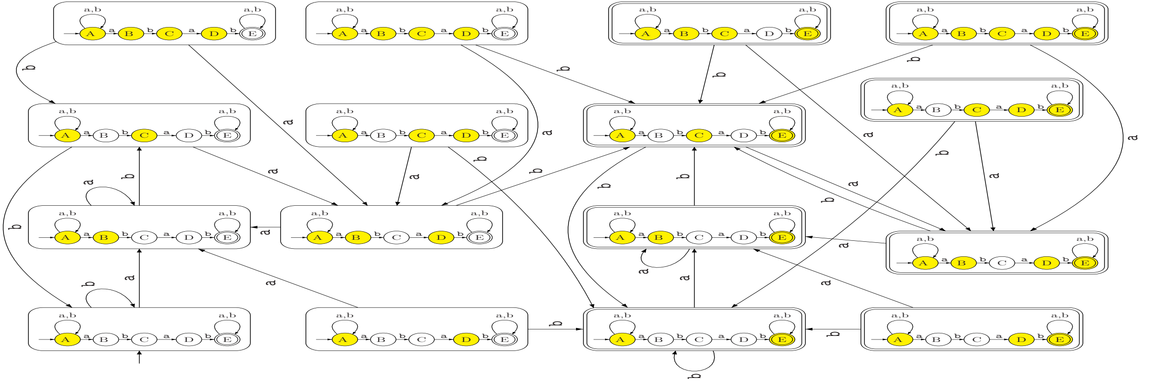
State space of the DFA is $\mathcal{P}(Q)$.

Example: DFA from NFA

NFA: (N1)



DFA:



Formal Tuple Notation for NFA

Definition 5.3.

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.

5.1.2
Algorithm for converting NFA to DFA

Recall I

Extending the transition function to strings

Definition 5.4.

For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ-transitions.

Definition 5.5.

Inductive definition of $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

- if $w = \varepsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$: $\delta^*(q, a) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \delta(p, a) \right)$
- if $w = ax$: $\delta^*(q, w) = \epsilon\text{reach}\left(\bigcup_{p \in \epsilon\text{reach}(q)} \bigcup_{r \in \delta^*(p,a)} \delta^*(r, x) \right)$

Recall II

Formal definition of language accepted by **N**

Definition 5.6.

A string w is accepted by NFA N if $\delta_N^*(s, w) \cap A \neq \emptyset$.

Definition 5.7.

The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$

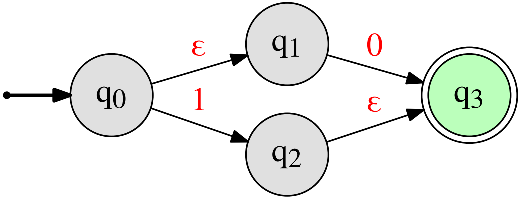
Subset Construction

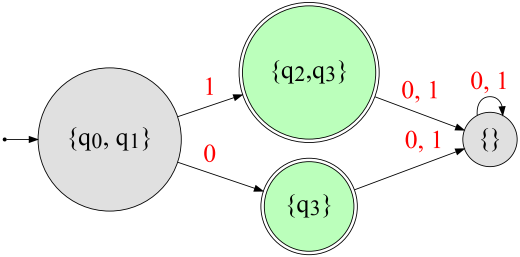
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $D = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$
- $A' = \{X \subseteq Q \mid X \cap A \neq \emptyset\}$
- $\delta'(X, a) = \cup_{q \in X} \delta^*(q, a)$ for each $X \subseteq Q$, $a \in \Sigma$.

Incremental construction

Only build states reachable from $s' = \epsilon\text{reach}(s)$ the start state of D





$\delta'(X, a) = \cup_{q \in X} \delta^*(q, a)$.

An optimization: Incremental algorithm

- Build D beginning with start state $s' == \epsilon\text{reach}(s)$
- For each existing state $X \subseteq Q$ consider each $a \in \Sigma$ and calculate the state $U = \delta'(X, a) = \cup_{q \in X} \delta^*(q, a)$ and add a transition.

  To compute $Z_{q,a} = \delta^*(q, a)$ - set of all states reached from q on character a
  - Compute $X_1 = \epsilon\text{reach}(q)$
  - Compute $Y_1 = \cup_{p \in X_1} \delta(p, a)$
  - Compute $Z_{q,a} = \epsilon\text{reach}(Y) = \cup_{r \in Y_1} \epsilon\text{reach}(r)$
- If U is a new state add it to reachable states that need to be explored.

5.1.3
Proof of correctness of conversion of NFA to DFA

Proof of Correctness

Theorem 5.8.

Let $N = (Q, \Sigma, s, \delta, A)$ *be a* NFA *and let* $D = (Q', \Sigma, \delta', s', A')$ *be a* DFA *constructed from* N *via the subset construction. Then* $L(N) = L(D)$.

Stronger claim:

Lemma 5.9.

For every string w, $\delta^*_N(s, w) = \delta^*_D(s', w)$.

Proof by induction on $|w|$.

Proof continued I

Lemma 5.10.

For every string w, $\delta_N^*(s, w) = \delta_D^*(s', w)$.

Proof:

Base case: $w = \epsilon$.

$\delta_N^*(s, \epsilon) = \epsilon\text{reach}(s)$.

$\delta_D^*(s', \epsilon) = s' = \epsilon\text{reach}(s)$ by definition of s'.

Proof continued II

Inductive hypothesis: $\forall w \in \mathbf{\Sigma}^*$, $|w| \leq k$: $\delta_N^*(s, w) = \delta_D^*(s', w)$.

Reminder: For a state $Y \subseteq Q(N)$ of the DFA D, we have (by definition):

$$\delta_D(Y, a) = \cup_{q \in Y} \delta_N^*(q, a)$$

Proof continued III

Lemma 5.11.

For every string w, $\delta_N^*(s, w) = \delta_D^*(s', w)$.

Inductive step: Consider any $w \in \Sigma^{k+1}$. $w = xa$ (Note: suffix definition of strings)

$\delta_N^*(s, xa) = \cup_{p \in \delta_N^*(s,x)} \delta_N^*(p, a)$ by recursive definition of δ_N^*

$\delta_D^*(s', xa) = \delta_D(\delta_D^*(s', x), a)$ by recursive definition of δ_D^*

By inductive hypothesis: $Y = \delta_N^*(s, x) = \delta_D^*(s', x)$.

$$\implies \delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) \qquad \text{and} \qquad \delta_D^*(s', xa) = \delta_D(Y, a).$$

By definition of δ_D: $\delta_D(Y, a) = \cup_{q \in Y} \delta_N^*(q, a)$.

$$\implies \delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) = \delta_D(Y, a) = \delta_D^*(s', xa).$$

$\square$

5.2
Closure Properties of Regular Languages

Regular Languages

Regular languages have three different characterizations

- Inductive definition via base cases and closure under union, concatenation and Kleene star
- Languages accepted by DFAs
- Languages accepted by NFAs

Regular language closed under many operations:

- union, concatenation, Kleene star via inductive definition or NFAs
- complement, union, intersection via DFAs
- homomorphism, inverse homomorphism, reverse, $\ldots$

Different representations allow for flexibility in proofs.

Example: PREFIX

Let L be a language over Σ.

Definition 5.1.

$\text{PREFIX}(L) = \{w \mid wx \in L, x \in \Sigma^*\}$

Theorem 5.2.

If L is regular then $\text{PREFIX}(L)$ is regular.

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L
$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$ $Y = \{q \in Q \mid q \text{ can reach some state in } A\}$
$Z = X \cap Y$
Create new DFA $M' = (Q, \Sigma, \delta, s, Z)$
Claim: $L(M') = \text{PREFIX}(L)$.

Exercise: SUFFIX

Let L be a language over Σ.

Definition 5.3.

$\text{SUFFIX}(L) = \{w \mid xw \in L, x \in \Sigma^*\}$

Prove the following:

Theorem 5.4.

If L is regular then $\mathit{PREFIX}(L)$ is regular.

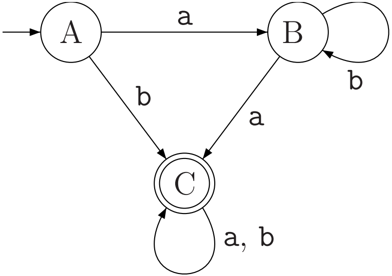
Exercise: SUFFIX

An alternative "proof" using a figure

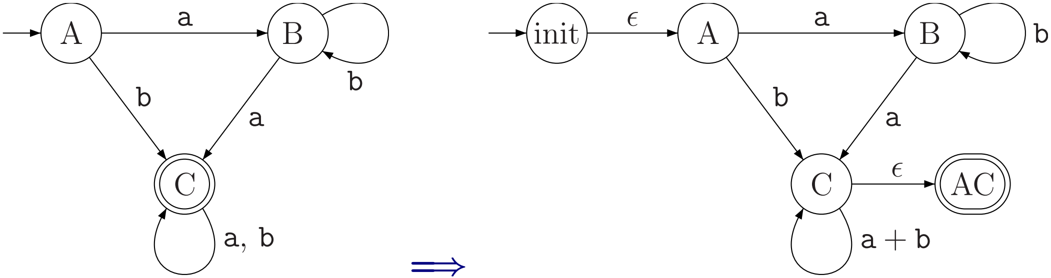
5.3

Algorithm for converting NFA into regular expression

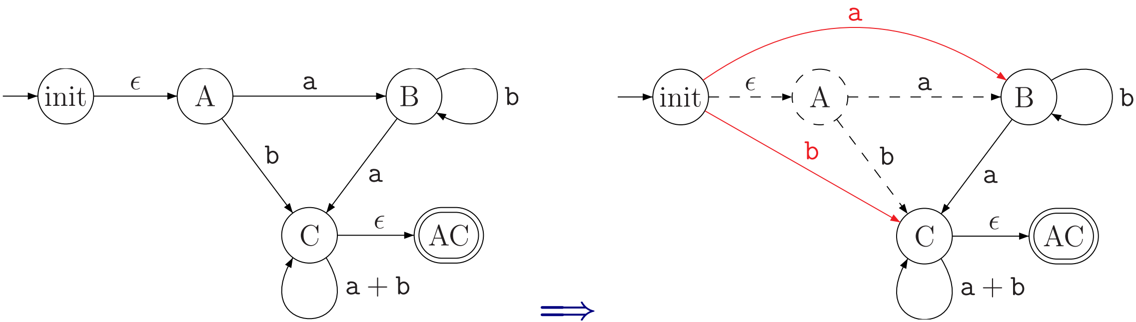
Stage 0: Input



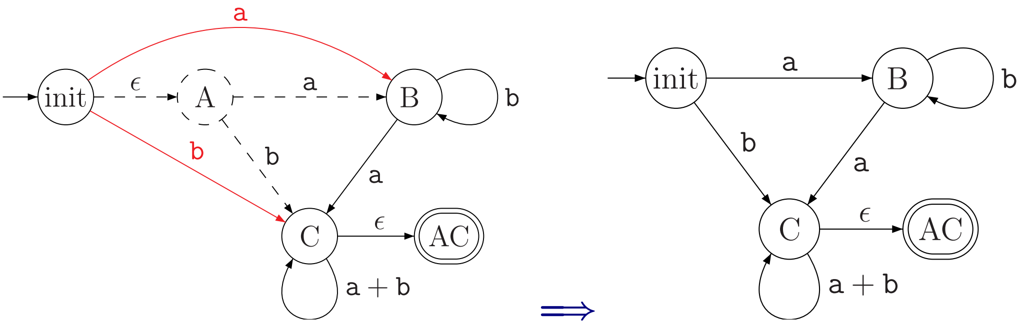
Stage 1: Normalizing



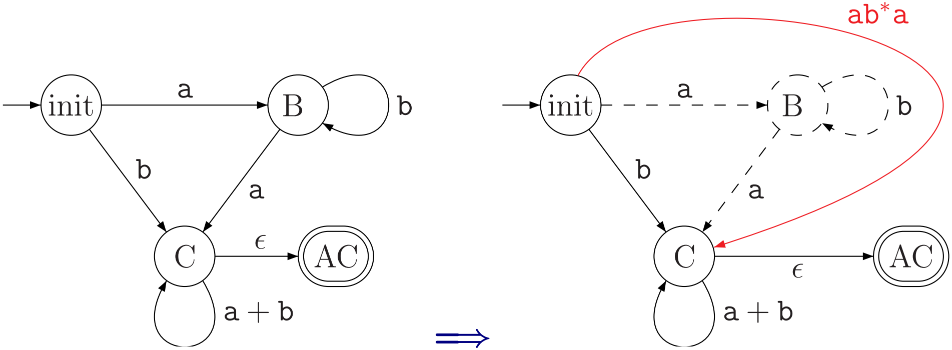
Stage 2: Remove state A



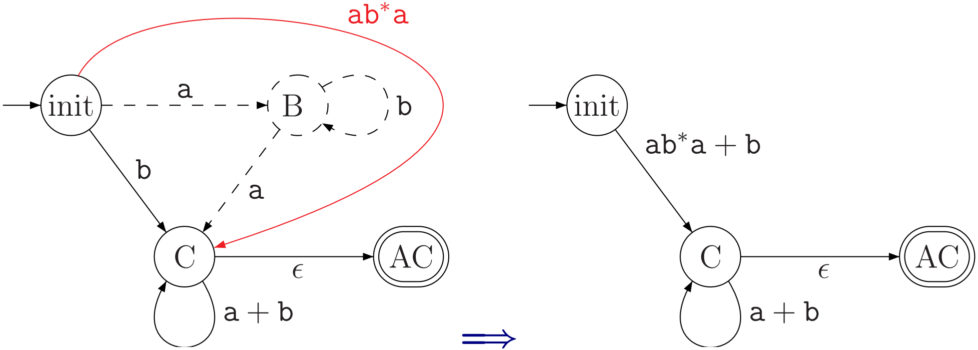
Stage 4: Redrawn without old edges



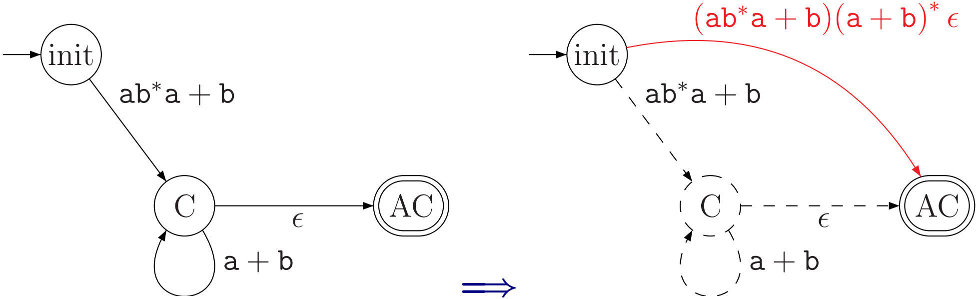
Stage 4: Removing B



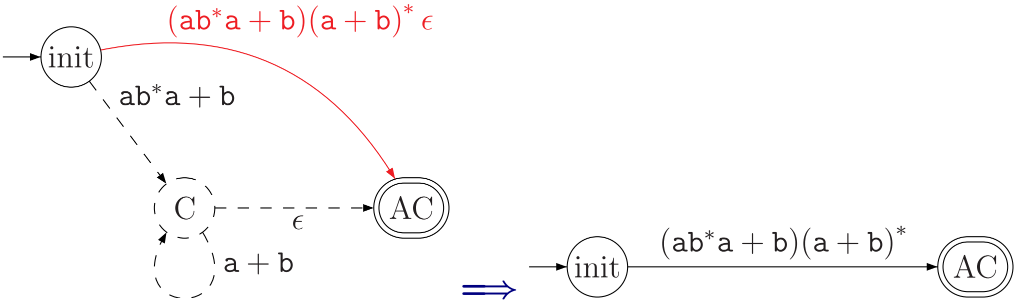
Stage 5: Redraw



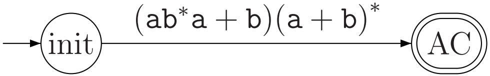
Stage 6: Removing C



Stage 7: Redraw



Stage 8: Extract regular expression



Thus, this automata is equivalent to the regular expression

$$(ab^*a + b)(a + b)^*.$$