Algorithms & Models of Computation
CS/ECE 374, Fall 2020

NFAs continued, Closure Properties of Regular Languages

Lecture 5

Tuesday, September 8, 2020

LaTeXed: September 8, 2020 12:38

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

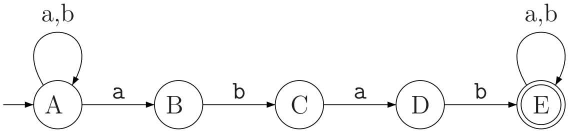
DFAs are memoryless...

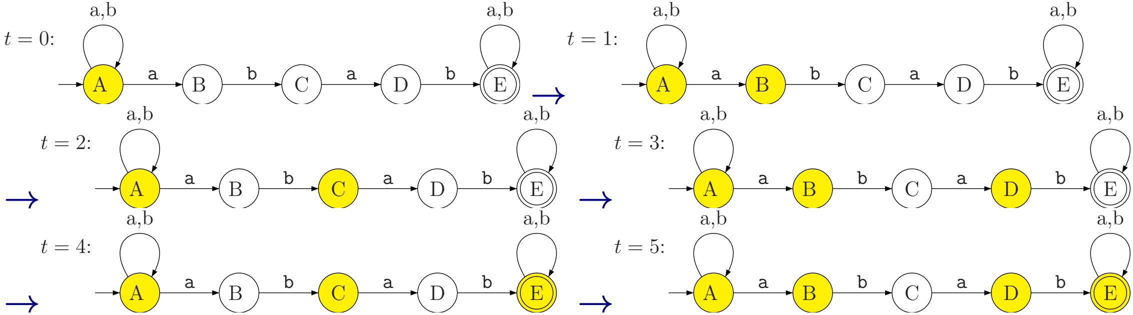
1. DFA knows only its current state.
2. The state is the memory.
3. To design a DFA, answer the question:
   What minimal info needed to solve problem.

Simulating NFA

Example the first revisited

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

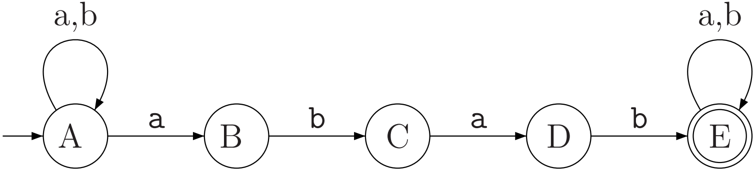




The state of the NFA

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.

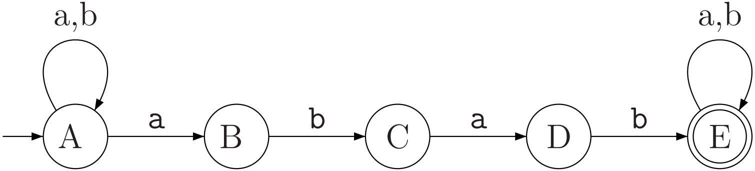
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

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.

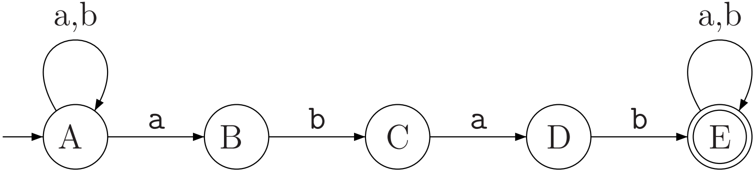
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

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.

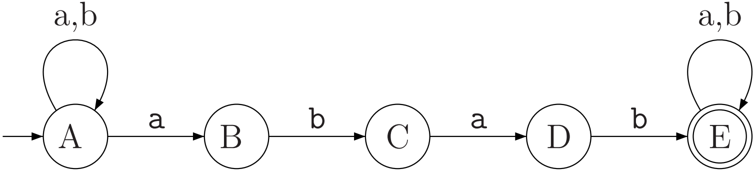
Possible configurations: $\emptyset$, $\{A\}$, $\{A, B\}$...

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)

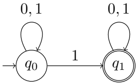


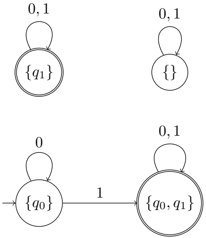
configuration: A set of states the automata might be in.

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

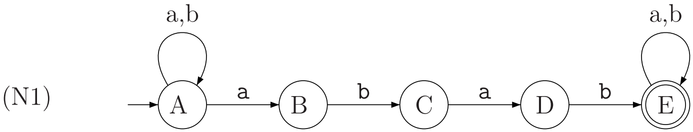
- Think of a program with fixed memory that needs to simulate NFA N on input w.
- What does it need to store after seeing a prefix x of w?
- It needs to know at least $\delta^*(s, x)$, the set of states that N could be in after reading x
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after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

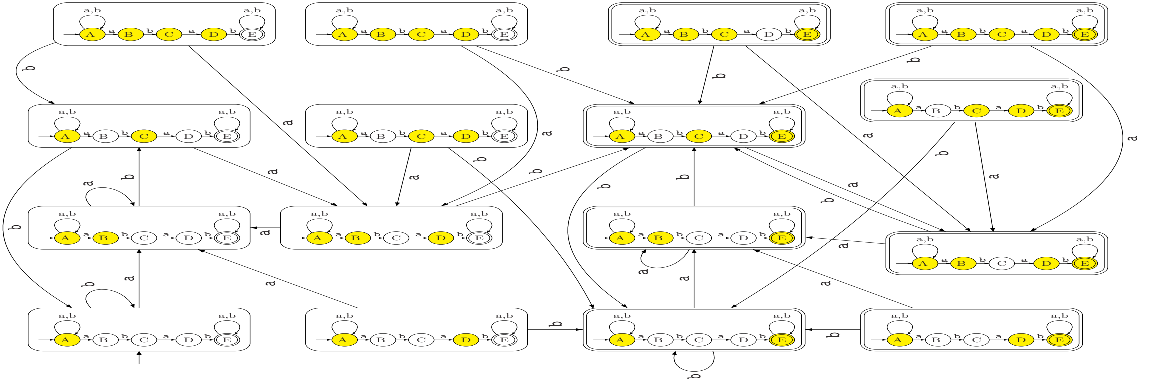
State space of the DFA is $\mathcal{P}(Q)$.

Example: DFA from NFA

NFA:



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)

5.1.2
Algorithm for converting NFA to DFA

Recall I

Extending the transition function to strings

Definition

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

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

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

Definition

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 $X \subseteq Q$, $a \in \Sigma$.

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 $X \subseteq Q$, $a \in \Sigma$.

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 $X \subseteq Q$, $a \in \Sigma$.

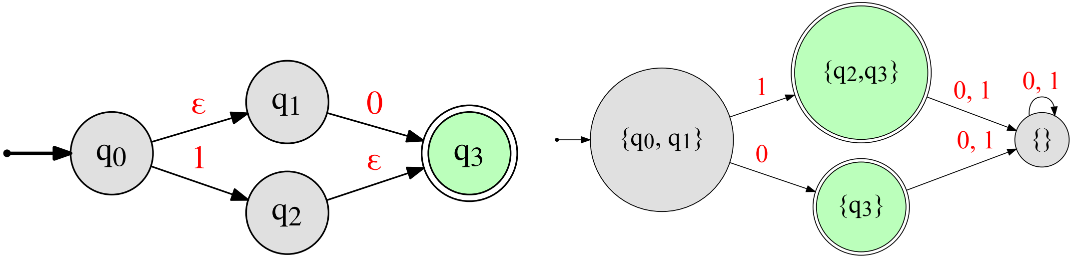
Subset Construction

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.

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.

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.

THE END

...

(for now)

5.1.3
Proof of correctness of conversion of NFA to DFA

Proof of Correctness

Theorem

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

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

Proof by induction on $|w|$.

Proof of Correctness

Theorem

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

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

Proof by induction on $|w|$.

Proof of Correctness

Theorem

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

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

Proof by induction on $|w|$.

Proof continued I

Lemma

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

Lemma

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

Inductive step: $\boldsymbol{w = xa}$ (Note: suffix definition of strings)

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

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

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

Thus $\delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) = \delta_D(Y, a)$ by definition of δ_D.

Therefore,

$\delta_N^*(s, xa) = \delta_D(Y, a) = \delta_D(\delta_D^*(s, x), a) = \delta_M^*(s', xa)$. which is what we need.

□

Proof continued II

Lemma

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

Inductive step: $w = xa$ (Note: suffix definition of strings)

$\delta^*_N(s, xa) = \cup_{p \in \delta^*_N(s,x)} \delta^*_N(p, a)$ by inductive definition of δ^*_N

$\delta^*_D(s', xa) = \delta_D(\delta^*_D(s, x), a)$ by inductive definition of δ^*_D

By inductive hypothesis: $Y = \delta^*_N(s, x) = \delta^*_D(s, x)$

Thus $\delta^*_N(s, xa) = \cup_{p \in Y} \delta^*_N(p, a) = \delta_D(Y, a)$ by definition of δ_D.

Therefore,

$\delta^*_N(s, xa) = \delta_D(Y, a) = \delta_D(\delta^*_D(s, x), a) = \delta^*_M(s', xa)$. which is what we need.

□

Proof continued II

Lemma

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

Inductive step: $w = xa$ (Note: suffix definition of strings)

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

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

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

Thus $\delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) = \delta_D(Y, a)$ by definition of δ_D.

Therefore,

$\delta_N^*(s, xa) = \delta_D(Y, a) = \delta_D(\delta_D^*(s, x), a) = \delta_M^*(s', xa)$. which is what we need.

□

Proof continued II

Lemma

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

Inductive step: $w = xa$ (Note: suffix definition of strings)

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

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

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

Thus $\delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) = \delta_D(Y, a)$ by definition of δ_D.

Therefore,

$\delta_N^*(s, xa) = \delta_D(Y, a) = \delta_D(\delta_D^*(s, x), a) = \delta_M^*(s', xa)$. which is what we need.

Proof continued II

Lemma

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

Inductive step: $w = xa$ (Note: suffix definition of strings)

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

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

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

Thus $\delta_N^*(s, xa) = \cup_{p \in Y} \delta_N^*(p, a) = \delta_D(Y, a)$ by definition of δ_D.

Therefore,
$\delta_N^*(s, xa) = \delta_D(Y, a) = \delta_D(\delta_D^*(s, x), a) = \delta_M^*(s', xa)$. which is what we need.

□

THE END

...

(for now)

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, ...

Different representations allow for flexibility in proofs.

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, ...

Different representations allow for flexibility in proofs.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

If L is regular then 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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

If L is regular then 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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

If L is regular then 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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

If L is regular then $PREFIX(L)$ 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: $L(M') = \text{PREFIX}(L)$.

Example: PREFIX

Let L be a language over Σ.

Definition

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

Theorem

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

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

Prove the following:

Theorem

If L is regular then $PREFIX(L)$ is regular.

Exercise: SUFFIX

An alternative "proof" using a figure

THE END

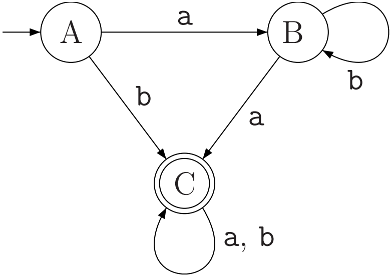
...

(for now)

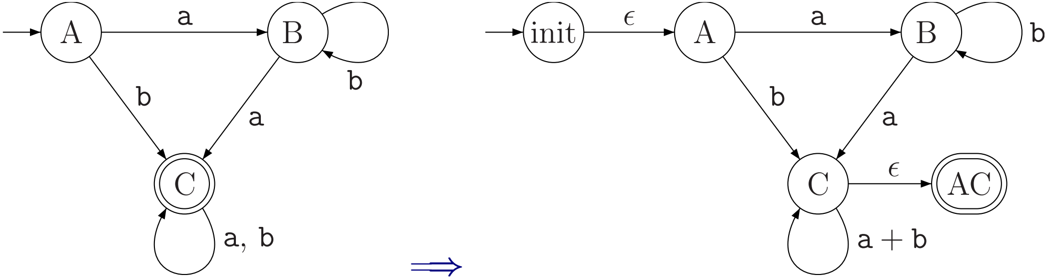
5.3

Algorithm for converting NFA into regular expression

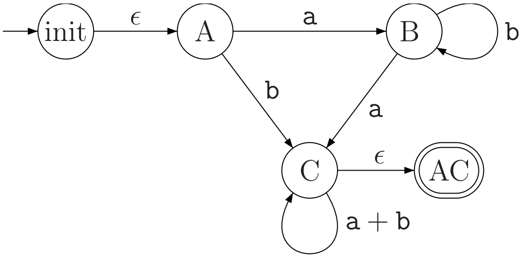
Stage 0: Input



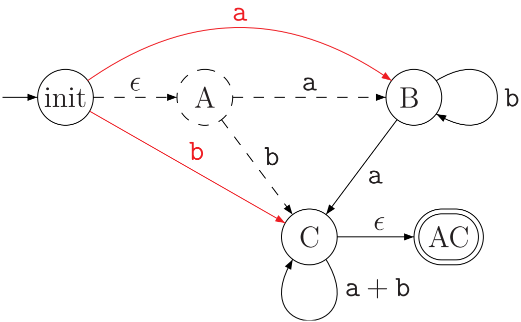
Stage 1: Normalizing



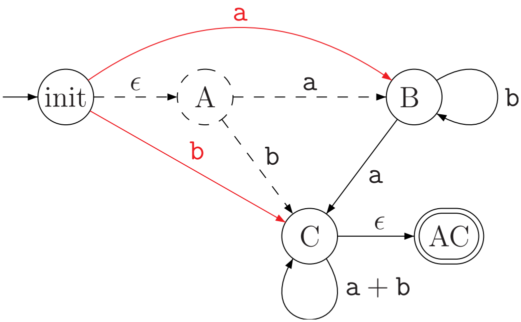
Stage 2: Remove state A



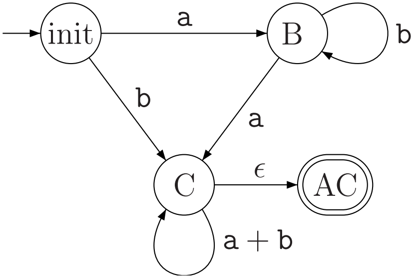
$\Longrightarrow$



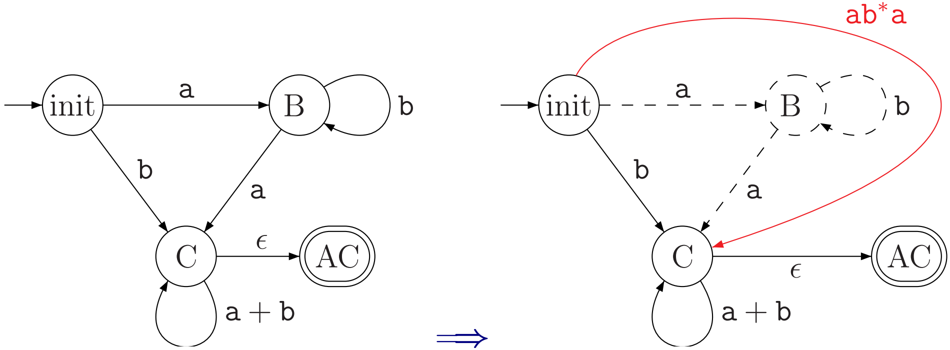
Stage 4: Redrawn without old edges



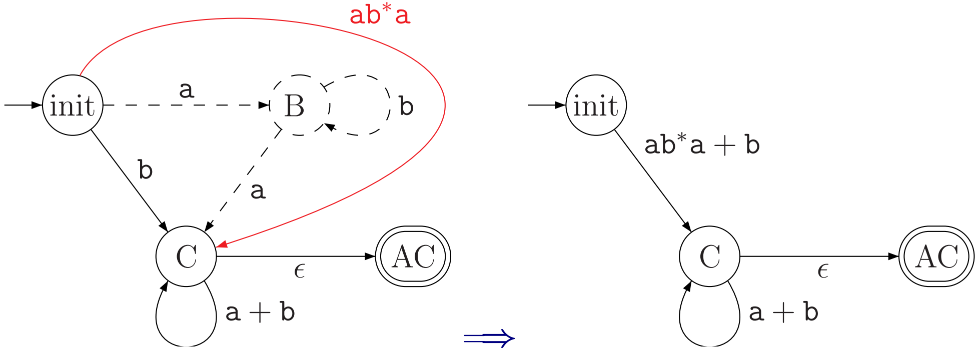
⟹



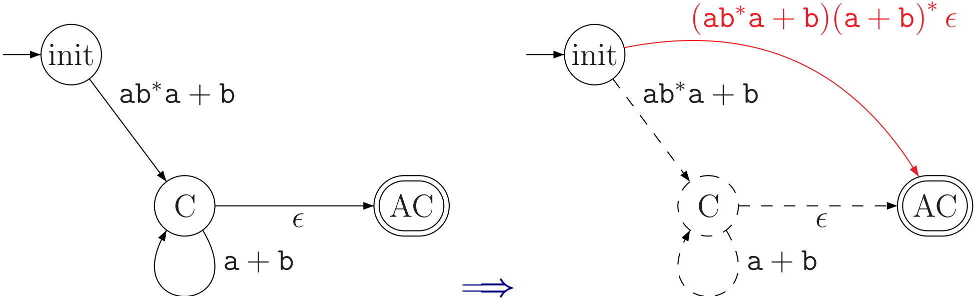
Stage 4: Removing B



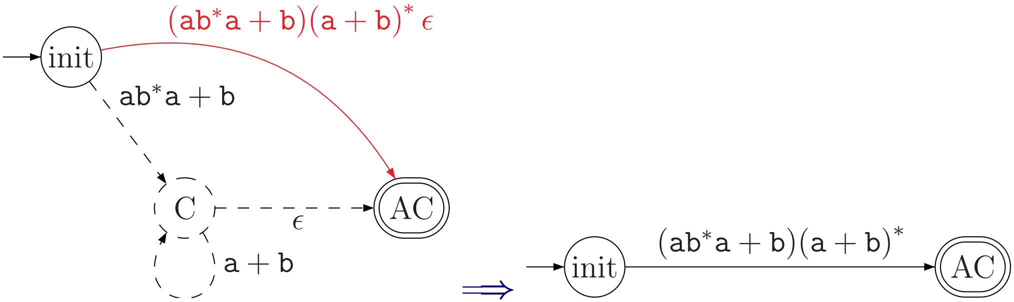
Stage 5: Redraw



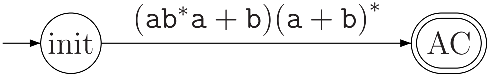
Stage 6: Removing C



Stage 7: Redraw



Stage 8: Extract regular expression



Thus, this automata is equivalent to the regular expression

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

THE END

...

(for now)