NFAs continued, Closure Properties of Regular Languages

Lecture 5

Tuesday, September 12, 2017

Regular Languages, DFAs, NFAs

Theorem

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

- DFAs are special cases of NFAs (trivial)
- NFAs accept regular expressions (we saw already)
- DFAs accept languages accepted by NFAs (today)
- 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 (trivial)
- NFAs accept regular expressions (we saw already)
- DFAs accept languages accepted by NFAs (today)
- Regular expressions for languages accepted by DFAs (later in the course)

Part I

Equivalence of NFAs and DFAs

Equivalence of NFAs and DFAs

Theorem

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

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.

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 $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$
  $\delta^*(q, a) = \cup_{p \in \epsilon\text{reach}(q)} (\cup_{r \in \delta(p,a)} \epsilon\text{reach}(r))$
- if $w = xa$,
  $\delta^*(q, w) = \cup_{p \in \delta^*(q,x)} (\cup_{r \in \delta(p,a)} \epsilon\text{reach}(r))$

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

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

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: A DFA M that simulates N should keep in its memory/state the set of states of N

Thus the state space of the DFA should be $\mathcal{P}(Q)$.

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: A DFA M that simulates N should keep in its memory/state the set of states of N

Thus the state space of the DFA should be $\mathcal{P}(Q)$.

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: A DFA M that simulates N should keep in its memory/state the set of states of N

Thus the state space of the DFA should be $\mathcal{P}(Q)$.

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: A DFA M that simulates N should keep in its memory/state the set of states of N

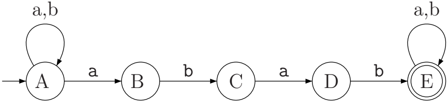
Thus the state space of the DFA should be $\mathcal{P}(Q)$.

Simulating NFA

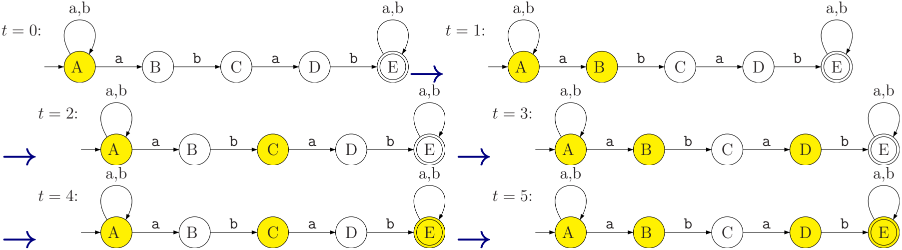
Example the first revisited

Previous lecture.. Ran

NFA$^{(N1)}$

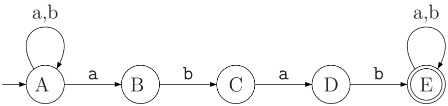


on input ***ababa***.

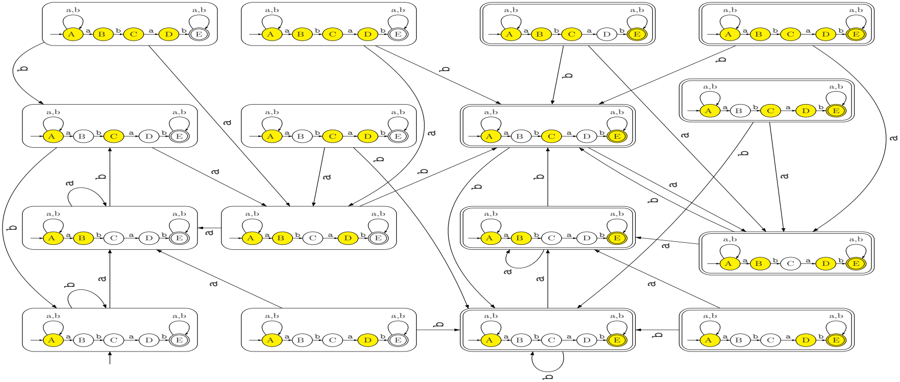


Example: DFA from NFA

NFA: (N1)



DFA:



Subset Construction

NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ 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 $M = (Q', \Sigma, \delta', s', A')$ 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 $M = (Q', \Sigma, \delta', s', A')$ 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 $X \subseteq Q$, $a \in \Sigma$.

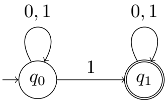
Subset Construction

NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $M = (Q', \Sigma, \delta', s', A')$ 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 $X \subseteq Q$, $a \in \Sigma$.

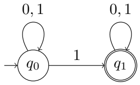
Example

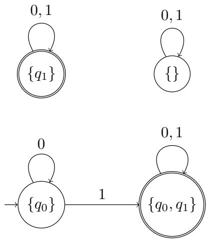
No ϵ-transitions



Example

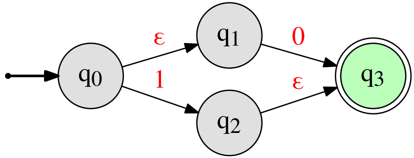
No ϵ-transitions

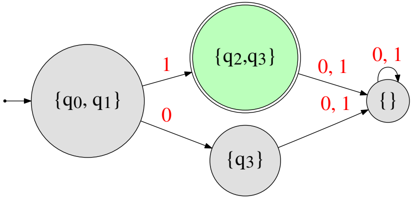




Incremental construction

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





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

Incremental algorithm

- Build M 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 $Y = \delta'(X, a) = \cup_{q \in X} \delta^*(q, a)$ and add a transition.
- If Y is a new state add it to reachable states that need to explored.

To compute $\delta^*(q, a)$ - set of all states reached from q on *string* a

- Compute $X = \epsilon\text{reach}(q)$
- Compute $Y = \cup_{p \in X} \delta(p, a)$
- Compute $Z = \epsilon\text{reach}(Y) = \cup_{r \in Y} \epsilon\text{reach}(r)$

Proof of Correctness

Theorem

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

Stronger claim:

Lemma

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

Proof by induction on $|w|$.

Base case: $w = \epsilon$.
$\delta^*_N(s, \epsilon) = \epsilon\text{reach}(s)$.
$\delta^*_M(s', \epsilon) = s' = \epsilon\text{reach}(s)$ by definition of s'.

Proof of Correctness

Theorem

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

Stronger claim:

Lemma

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

Proof by induction on $|w|$.

Base case: $w = \epsilon$.
$\delta^*_N(s, \epsilon) = \epsilon\text{reach}(s)$.
$\delta^*_M(s', \epsilon) = s' = \epsilon\text{reach}(s)$ by definition of s'.

Proof continued

Lemma

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

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

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

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

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

Therefore,

$\delta^*_N(s, xa) = \delta_M(Y, a) = \delta_M(\delta^*_M(s, x), a) = \delta^*_M(s', xa)$

which is what we need.

Proof continued

Lemma

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

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

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

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

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

Therefore,

$\delta^*_N(s, xa) = \delta_M(Y, a) = \delta_M(\delta^*_M(s, x), a) = \delta^*_M(s', xa)$

which is what we need.

Proof continued

Lemma

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

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

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

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

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

Proof continued

Lemma

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

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^*

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

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

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

Therefore,

$\delta_N^*(s, xa) = \delta_M(Y, a) = \delta_M(\delta_M^*(s, x), a) = \delta_M^*(s', xa)$

which is what we need.

Proof continued

Lemma

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

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

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

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

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

Part II

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

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

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

Theorem

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

Exercise: SUFFIX

Let L be a language over Σ.

Definition

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

Prove the following:

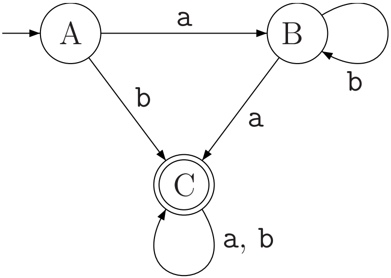
Theorem

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

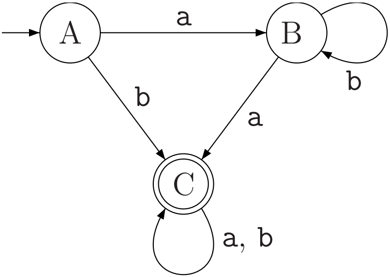
Part III

Regex to NFA

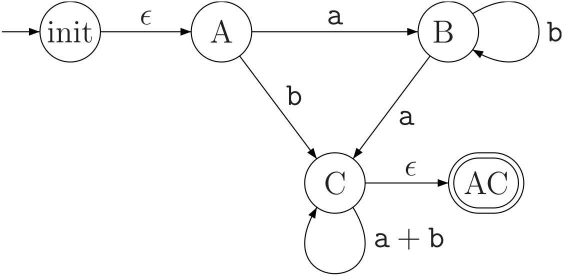
Stage 0: Input



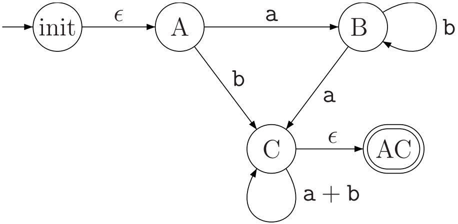
Stage 1: Normalizing

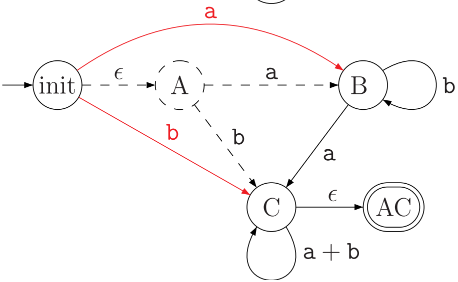


2: Normalizing it.

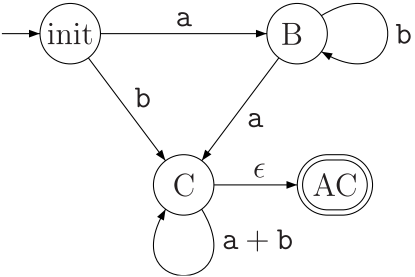


Stage 2: Remove state A

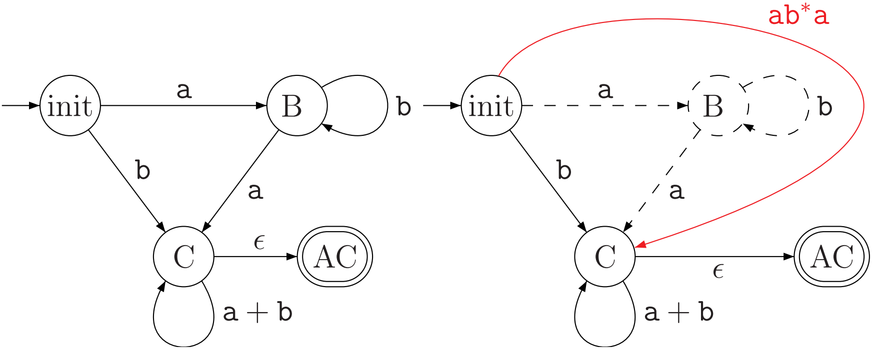




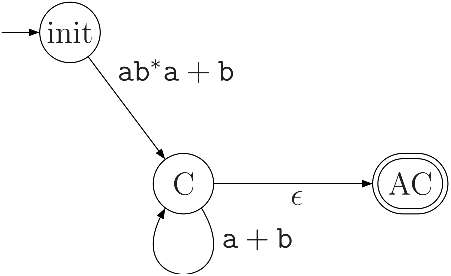
Stage 4: Redrawn without old edges



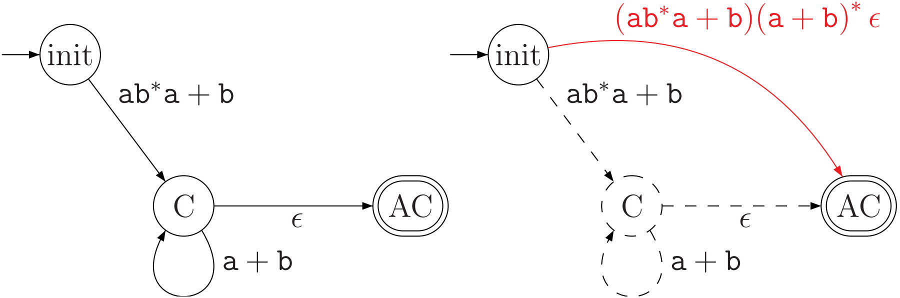
Stage 4: Removing B



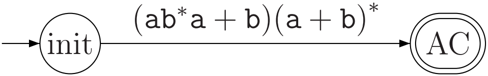
Stage 5: Redraw



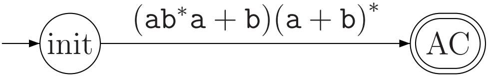
Stage 6: Removing C



Stage 7: Redraw



Stage 8: Extract regular expression



Thus, this automata is equivalent to the regular expression $\boldsymbol{(ab^*a + b)(a + b)^*}$.