NFAs continued, Closure Properties of Regular Languages

Lecture 5

Tuesday, September 6, 2022

LaTeXed: October 13, 2022 14:17

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 $\mathbf{N}$ *there is a DFA* $\mathbf{M}$ *such that* $\mathbf{L(M) = L(N)}$.

5.1.1
The idea of the conversion of NFA to DFA

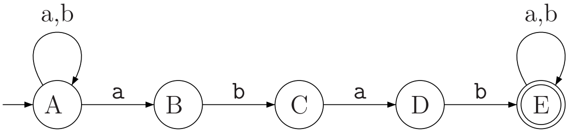
DFAs are memoryless...

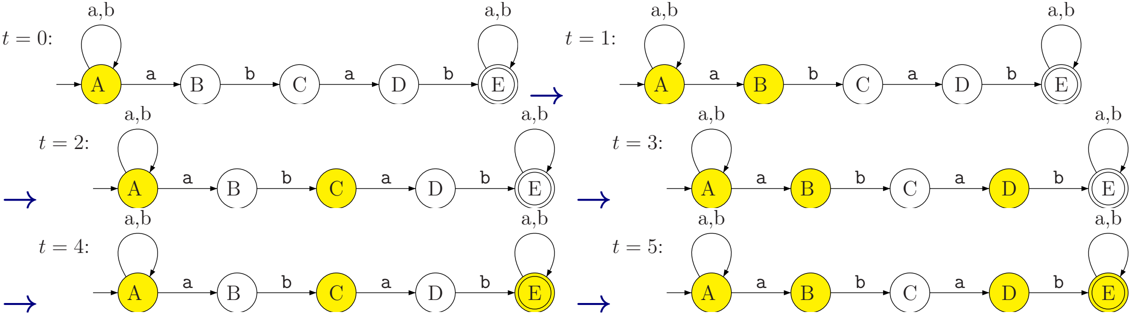
1. DFA knows only its current state.
2. The state is the memory.
3. To design a DFA, answer the question:
   What minimal info needed to solve problem.

Simulating NFA

Example the first revisited

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

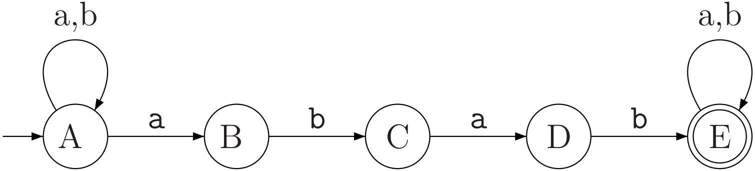




The state of the NFA

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

(N1)



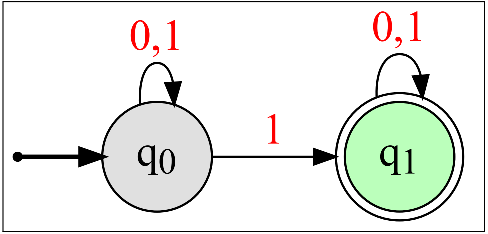
configuration: A set of states the automata might be in.

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

Big idea: Build a DFA on the configurations.

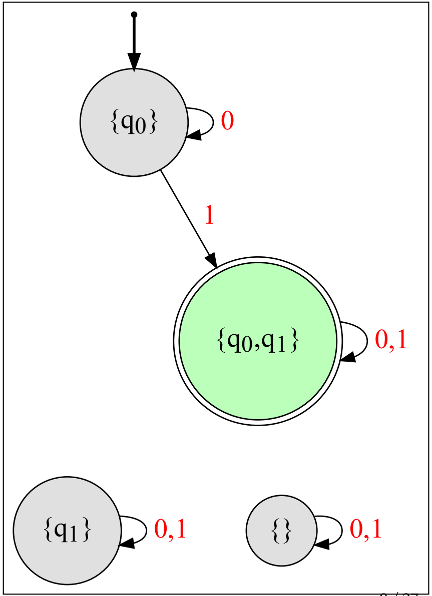
Example: Subset construction

DFA:



$\Longrightarrow$

NFA:



Simulating an NFA by a DFA

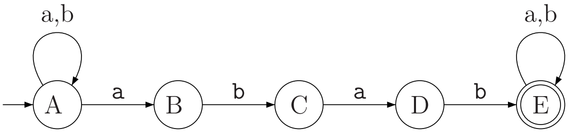
- Think of a program with fixed memory that needs to simulate NFA $\mathbf{N}$ on input $\mathbf{w}$.
- What does it need to store after seeing a prefix $\mathbf{x}$ of $\mathbf{w}$?
- It needs to know at least $\delta^*(\mathbf{s}, \mathbf{x})$, the set of states that $\mathbf{N}$ could be in after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

Key Observation: DFA $\mathbf{M}$ simulating $\mathbf{N}$ should know current configuration of $\mathbf{N}$.

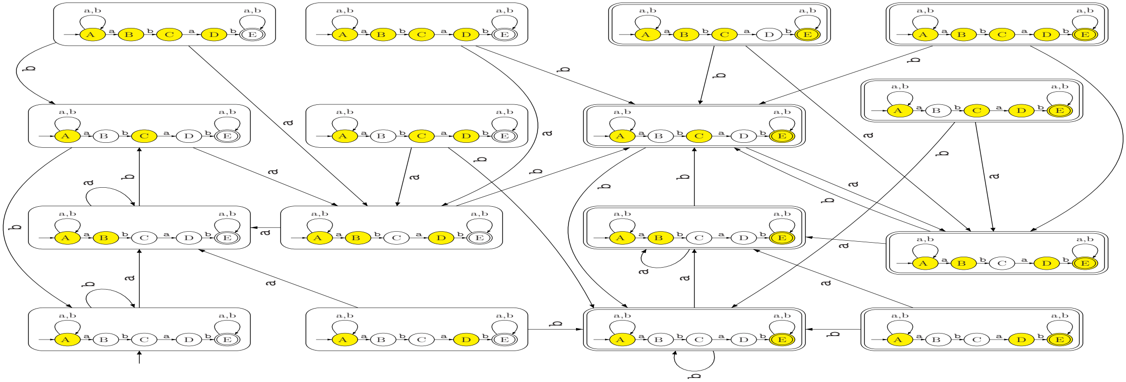
State space of the DFA is $\mathcal{P}(\mathbf{Q})$.

Example: DFA from NFA

NFA: (N1)



DFA:



Formal Tuple Notation for NFA

Definition 5.3.

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

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

$\delta(\mathbf{q}, \mathbf{a})$ for $\mathbf{a} \in \mathbf{\Sigma} \cup \{\epsilon\}$ is a subset of $\mathbf{Q}$ — 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 $\mathbf{w}$ is accepted by NFA $\mathbf{N}$ if $\delta^*_{\mathbf{N}}(\mathbf{s}, \mathbf{w}) \cap \mathbf{A} \neq \emptyset$.

Definition 5.7.

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

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

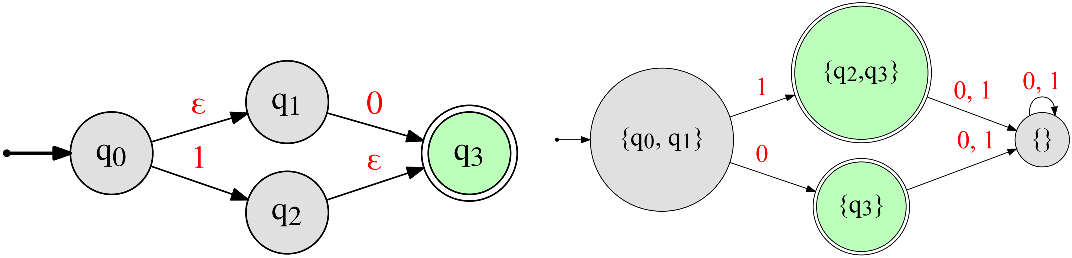
Subset Construction

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

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

Incremental construction

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



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

An optimization: Incremental algorithm

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

  To compute $\mathbf{Z_{q,a} = \delta^*(q, a)}$ - set of all states reached from $\mathbf{q}$ on character $\mathbf{a}$
  - Compute $\mathbf{X_1 = \epsilon reach(q)}$
  - Compute $\mathbf{Y_1 = \cup_{p \in X_1} \delta(p, a)}$
  - Compute $\mathbf{Z_{q,a} = \epsilon reach(Y) = \cup_{r \in Y_1} \epsilon reach(r)}$
- If $\mathbf{U}$ 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 $\mathbf{N} = (\mathbf{Q}, \mathbf{\Sigma}, \mathbf{s}, \delta, \mathbf{A})$ *be a* NFA *and let* $\mathbf{D} = (\mathbf{Q}', \mathbf{\Sigma}, \delta', \mathbf{s}', \mathbf{A}')$ *be a* DFA *constructed from* $\mathbf{N}$ *via the subset construction. Then* $\mathbf{L}(\mathbf{N}) = \mathbf{L}(\mathbf{D})$.

Stronger claim:

Lemma 5.9.

For every string $\mathbf{w}$, $\delta^*_{\mathbf{N}}(\mathbf{s}, \mathbf{w}) = \delta^*_{\mathbf{D}}(\mathbf{s}', \mathbf{w})$.

Proof by induction on $|\mathbf{w}|$.

Proof continued I

Lemma 5.10.

For every string $\mathbf{w}$, $\delta_{\mathbf{N}}^{*}(\mathbf{s}, \mathbf{w}) = \delta_{\mathbf{D}}^{*}(\mathbf{s}', \mathbf{w})$.

Proof:

Base case: $\mathbf{w} = \epsilon$.

$\delta_{\mathbf{N}}^{*}(\mathbf{s}, \epsilon) = \epsilon\text{reach}(\mathbf{s})$.

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

Proof continued II

Lemma 5.11.

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

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.

□

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.

Example: PREFIX

Let $\mathbf{L}$ be a language over $\mathbf{\Sigma}$.

Definition 5.1.

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

Theorem 5.2.

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

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

Exercise: SUFFIX

Let $\mathbf{L}$ be a language over $\mathbf{\Sigma}$.

Definition 5.3.

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

Prove the following:

Theorem 5.4.

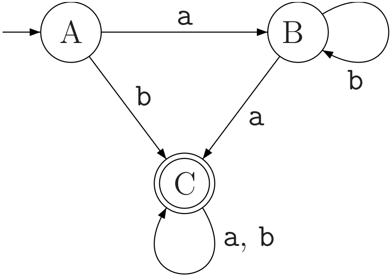
If $\mathbf{L}$ is regular then PREFIX$(\mathbf{L})$ is regular.

Exercise: SUFFIX

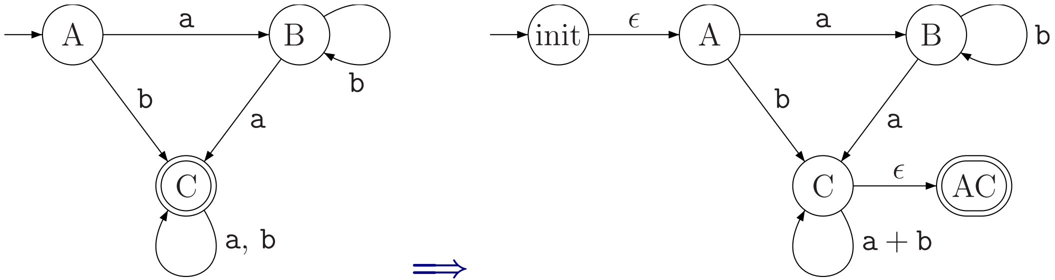
An alternative "proof" using a figure

5.3
Algorithm for converting NFA into regular expression

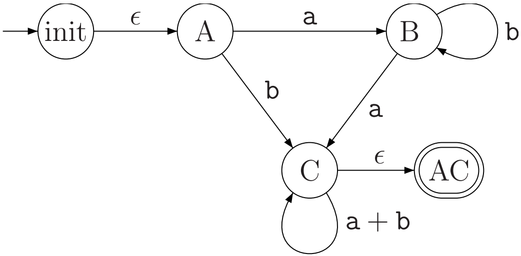
Stage 0: Input



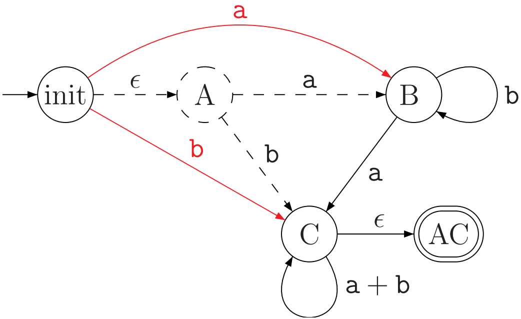
Stage 1: Normalizing



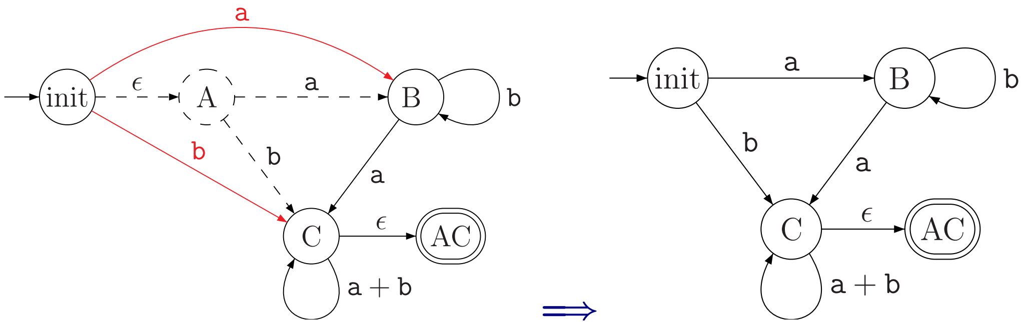
Stage 2: Remove state A



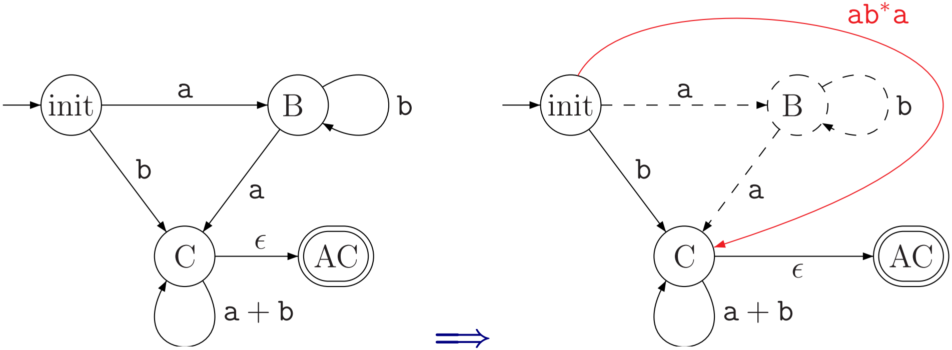
⟹



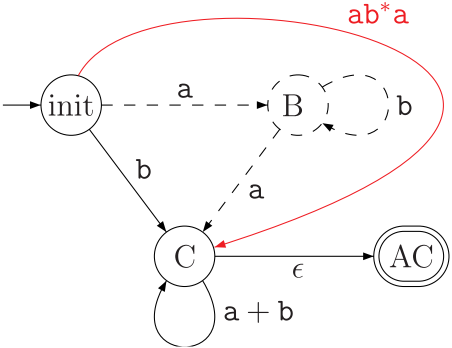
Stage 4: Redrawn without old edges



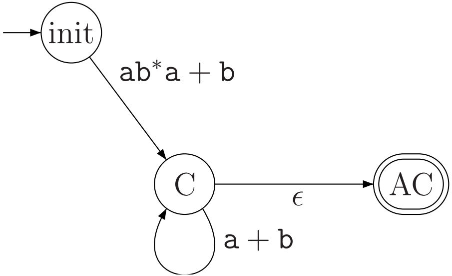
Stage 4: Removing B



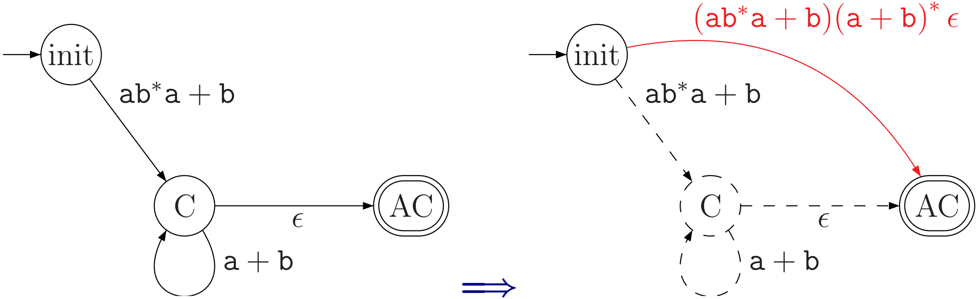
Stage 5: Redraw



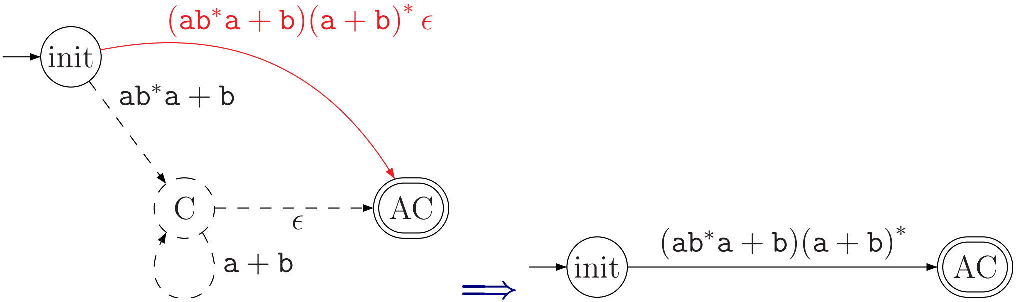
$\Longrightarrow$



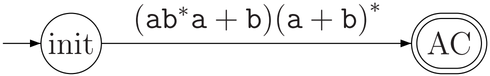
Stage 6: Removing C



Stage 7: Redraw



Stage 8: Extract regular expression



Thus, this automata is equivalent to the regular expression

$$\mathbf{(ab^*a + b)(a + b)^*.}$$