NFAs continued, Closure Properties of Regular Languages

Lecture 5

January 31, 2023

Regular Languages, DFAs, NFAs

Theorem

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

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 (today, informally)

Part I

Equivalence of NFAs and DFAs

Equivalence of NFAs and DFAs

Theorem

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

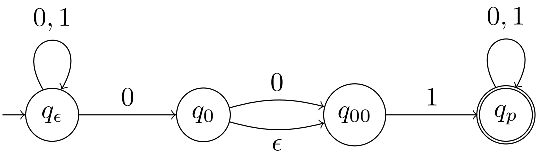
Equivalence of NFAs and DFAs

Theorem

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

- The number of states in N can be exponential in number of states of M
- Examples show that it is necessary in some cases. That is, there are regular languages for which the best/smallest DFA has exponentially more states than the best/smallest NFA.

NFAs and acceptance



A NFA N accepts a string w iff some accepting state is reached by N from the start state on input w.

The language accepted (or recognized) by a NFA N is denote by $L(N)$ and defined as: $L(N) = \{w \mid N \text{ accepts } w\}$.

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 susbet 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 $\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = a$ where $a \in \Sigma$
  $\delta^*(q, a) = \cup_{p \in \epsilon\text{reach}(q)} \left(\cup_{r \in \delta(p,a)} \epsilon\text{reach}(r)\right)$
- if $w = ax$,
  $\delta^*(q, w) = \cup_{p \in \delta^*(q,a)} \delta^*(p, x)$
- if $w = xa$, alternate definition based on string suffixes
  $\delta^*(q, w) = \cup_{p \in \delta^*(q,r)} \delta^*(p, a)$

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

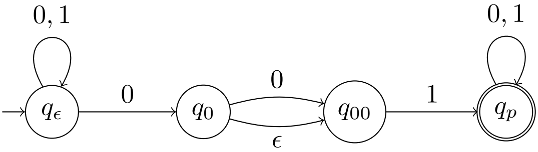
$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$

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?
- **Easier question:** Can we write a program that decides whether N accepts a string?

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?
- **Easier question:** Can we write a program that decides whether N accepts a string?



Does N accept 00010101001000100001000001000000111110001?

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?

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?

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 $\mathcal{P}(Q)$.

Subset Construction

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

- $Q' = \mathcal{P}(Q)$

Subset Construction

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

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$

Subset Construction

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

- $Q' = \mathcal{P}(Q)$
- $s' = \epsilon\text{reach}(s) = \delta^*(s, \epsilon)$
- $A' = \{X \subseteq Q \mid X \cap A \neq \emptyset\}$

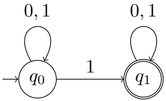
Subset Construction

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

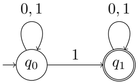
Example

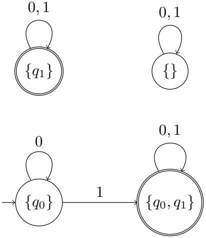
No ϵ-transitions



Example

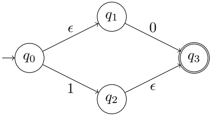
No ϵ-transitions

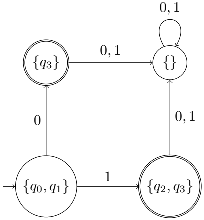




Incremental construction

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* $L(N) = L(M)$.

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 defn of δ^*_N

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 defn of δ^*_N

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

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 defn of δ_N^*

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

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

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 defn of δ^*_N

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

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

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

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 defn of δ_N^*

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

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

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

DFA/NFA to Regular Expressions

DFA to Regular Expressions

Theorem

Given a DFA $M = (Q, \Sigma, \delta, s, A)$ there is a regular expression r such that $L(r) = L(M)$. That is, regular expressions are as powerful as DFAs (and hence also NFAs).

- Simple algorithm but formal proof is involved. See notes.
- An easier proof via a more involved algorithm (maybe later in the course)

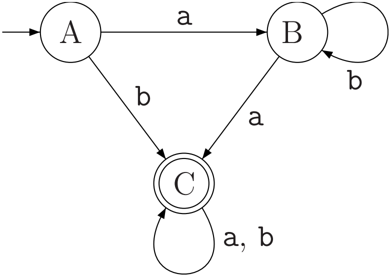
NFA to Regular Expressions

In fact the algorithm transforms an NFA $N = (Q, \Sigma, \delta, s, A)$ to a regular expression via GNFAs which are generalized NFAs.

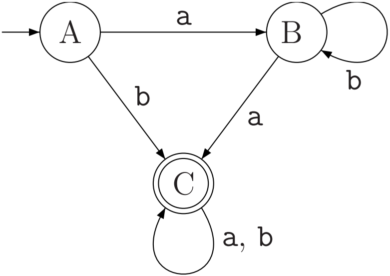
Informal Definition: A generalized NFA or a GNFA is specified like an NFA but each arc is labeled with a regular expression. One can transition an arc (p, q) from state p to state q labeled with a regular expression r by reading any string $w \in L(r)$.

One can show that GNFAs are equivalent to NFAs by simply replacing each arc with reg exp r via a NFA for r via algorithm from last semester.

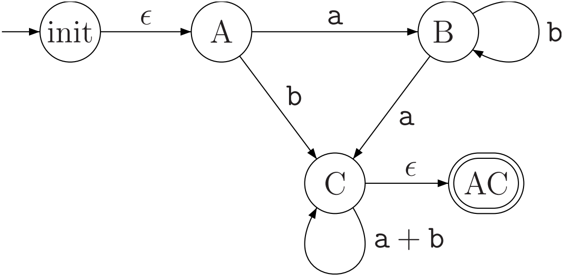
Stage 0: Input



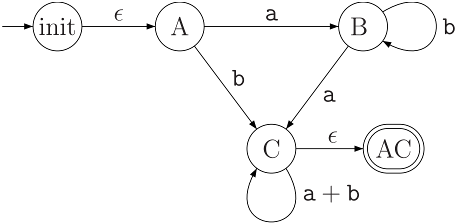
Stage 1: Normalizing

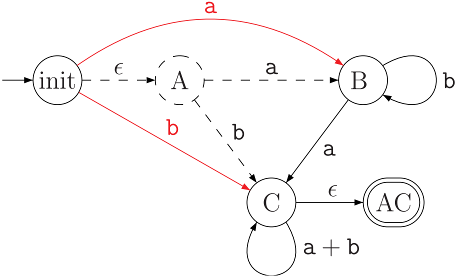


2: Normalizing it.

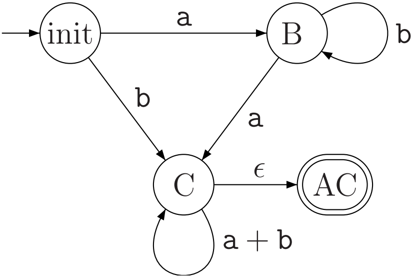


Stage 2: Remove state A

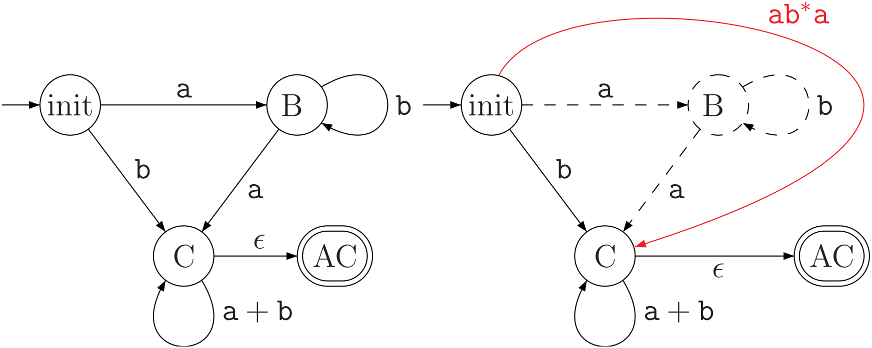




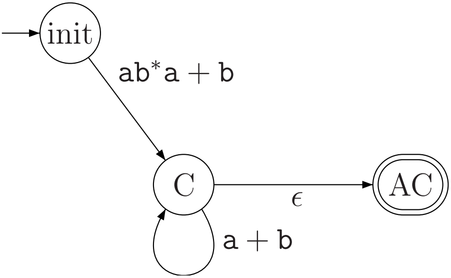
Stage 4: Redrawn without old edges



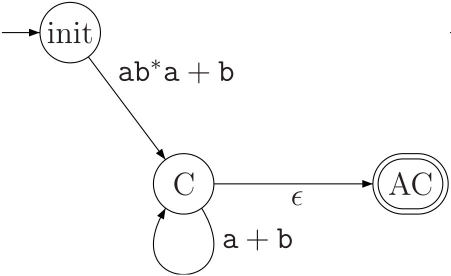
Stage 4: Removing B

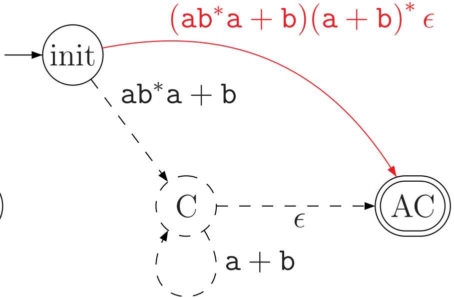


Stage 5: Redraw

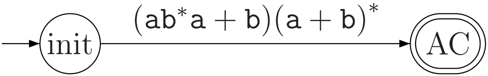


Stage 6: Removing C





Stage 7: Redraw



Stage 8: Extract regular expression

$$\xrightarrow{}\ \text{init} \xrightarrow{(\mathtt{ab}^*\mathtt{a}+\mathtt{b})(\mathtt{a}+\mathtt{b})^*} \text{AC}$$

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

Part III

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 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

Examples: PREFIX and SUFFIX

Let L be a language over Σ.

Definition

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

Definition

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

Examples: PREFIX and SUFFIX

Let L be a language over Σ.

Definition

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

Definition

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

Theorem

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

Theorem

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

PREFIX

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L

Create new DFA/NFA to accept PREFIX(L) (or SUFFIX(L)).

PREFIX

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L

Create new DFA/NFA to accept PREFIX(L) (or SUFFIX(L)).

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$
$Y = \{q \in Q \mid q \text{ can reach some state in } A\}$
$Z = X \cap Y$

Theorem

Consider DFA $M' = (Q, \Sigma, \delta, s, Z)$. $L(M') = PREFIX(L)$.

SUFFIX

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$

SUFFIX

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$

Consider NFA $N = (Q \cup \{s'\}, \Sigma, \delta', s', A)$. Add new start state s' and ϵ-transition from s' to each state in X.

SUFFIX

Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA that recognizes L

$X = \{q \in Q \mid s \text{ can reach } q \text{ in } M\}$

Consider NFA $N = (Q \cup \{s'\}, \Sigma, \delta', s', A)$. Add new start state s' and ϵ-transition from s' to each state in X.

Claim: $L(N) = \text{SUFFIX}(L)$.