Non-deterministic Finite Automata (NFAs)

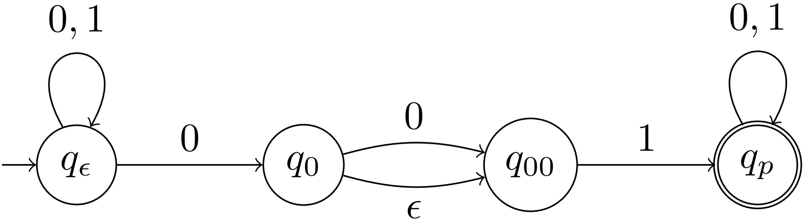
Lecture 4

January 30, 2025

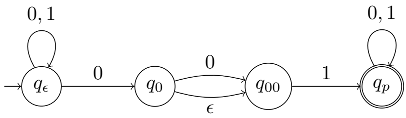
Part I

NFA Introduction

A Strange DFA

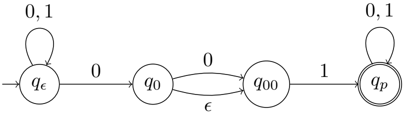


NFAs, Informally



NFAs are DFAs with extra flexibility:

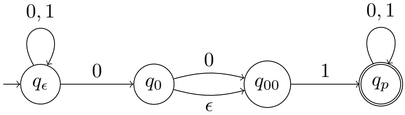
NFAs, Informally



NFAs are DFAs with extra flexibility:

DFA	NFA
Can only change state when reading a character	Allowed to move without a character (ϵ-transitions)

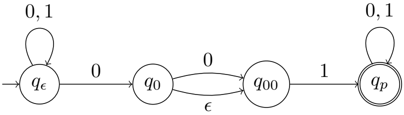
NFAs, Informally



NFAs are DFAs with extra flexibility:

DFA	**NFA**
Can only change state when reading a character	Allowed to move without a character (ϵ-transitions)
Given state and character read, moves to a particular state	Define a set of *possible* states to move to

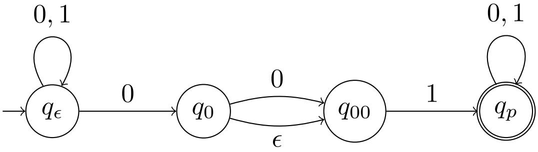
NFAs, Informally



NFAs are DFAs with extra flexibility:

DFA	NFA
Can only change state when reading a character	Allowed to move without a character (ϵ-transitions)
Given state and character read, moves to a particular state	Define a set of *possible* states to move to
Accepts a string if it leads to an accepting state	Accepts a string if *possible* to lead to an accepting state

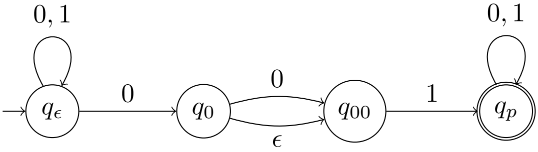
NFA Acceptance Examples



Does this NFA accept

- 0010?

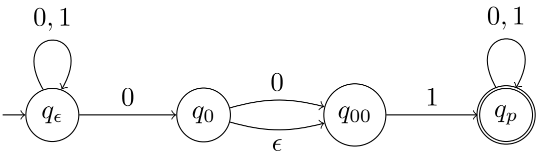
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!

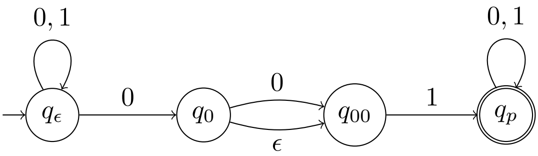
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01?

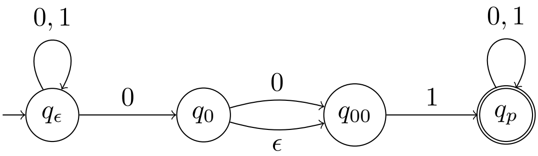
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!

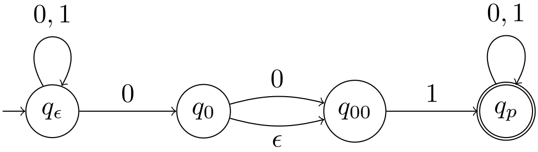
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010?

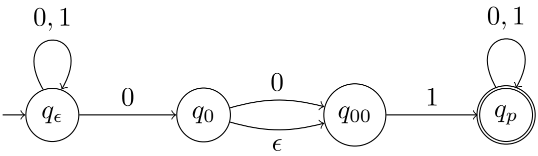
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!

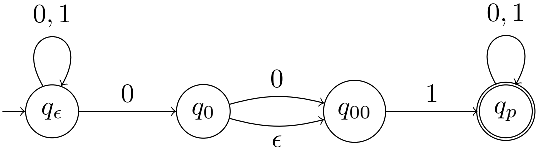
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!
- 111?

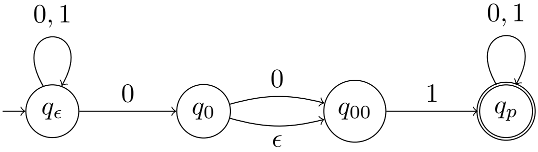
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!
- 111? Nope!

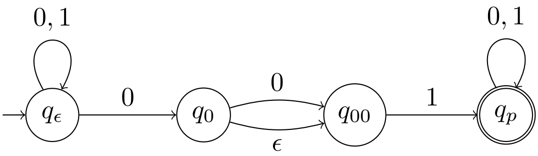
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!
- 111? Nope!
- 100?

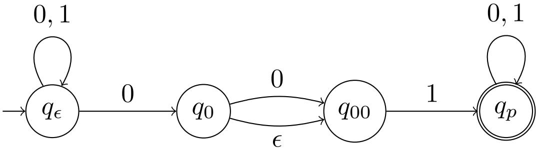
NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!
- 111? Nope!
- 100? Nope!

NFA Acceptance Examples



Does this NFA accept

- 0010? Yes!
- 01? Yes!
- 1010? Yes!
- 111? Nope!
- 100? Nope!

Note: Arguing that an NFA *doesn't* accept a string is tricky–showing that it *can* reach a rejecting state is insufficient!

Formal Definition

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.

Formal Definition

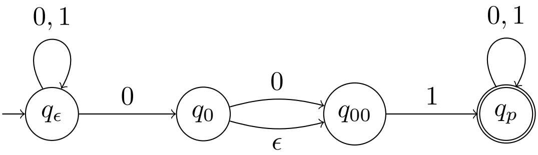
Definition

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.

Key differences from DFA: δ can take ϵ as input, and outputs a *set* of states instead of a single state.

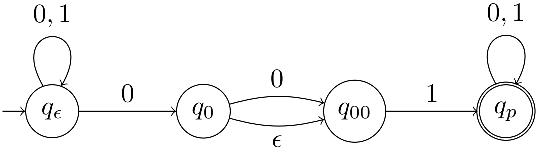
Example



- $Q =$
- $\Sigma =$
- $s =$
- $A =$

- $\delta(q, a) =$

Example



- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- $s = q_\epsilon$
- $A = \{q_p\}$
- $\delta(q, a) = \begin{cases} \{q_\epsilon, q_0\} & \text{if } q = q_\epsilon, a = 0 \\ \emptyset & \text{if } q = q_0, a = 1 \\ \{q_{00}\} & \text{if } q = q_0, a = \epsilon \\ \dots \end{cases}$

As with DFAs, the drawing and the tuple define the same object–use whichever one is easier in context.

Defining Acceptance

Informally, we said an NFA accepts a string w if it is possible to reach an accepting state when reading w.

Defining Acceptance

Informally, we said an NFA accepts a string w if it is possible to reach an accepting state when reading w.

To formalize this, want to define $\delta^*(q, w)$ as the set of possible states we could get to starting from q and reading w.

Defining Acceptance

Informally, we said an NFA accepts a string w if it is possible to reach an accepting state when reading w.

To formalize this, want to define $\delta^*(q, w)$ as the set of possible states we could get to starting from q and reading w.

First attempt:

- If $w = \epsilon$, $\delta^*(q, w) =$
- If $w = ax$, $\delta^*(q, w) =$

Defining Acceptance

Informally, we said an NFA accepts a string w if it is possible to reach an accepting state when reading w.

To formalize this, want to define $\delta^*(q, w)$ as the set of possible states we could get to starting from q and reading w.

First attempt:

- If $w = \epsilon$, $\delta^*(q, w) = \{q\}$
- If $w = ax$, $\delta^*(q, w) = \cup_{r \in \delta(q,a)} \delta^*(r, x)$

This doesn't allow for ϵ-transitions!

A Helpful Definition

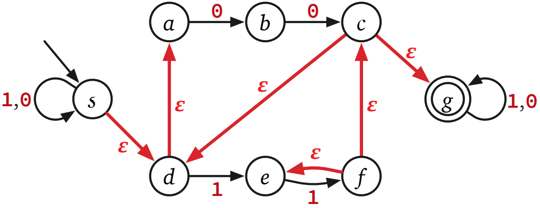
Definition

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.

A Helpful Definition

Definition

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



- ϵreach$(s) =$
- ϵreach$(b) =$
- ϵreach$(f) =$

Formally Defining δ^*

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

For NFA $N = (Q, \Sigma, \delta, s, A)$, δ^* is a function from $Q \times \Sigma^*$ to $\mathcal{P}(Q)$ defined by

- if $w = \epsilon$, $\delta^*(q, w) =$
- if $w = ax$, $\delta^*(q, w) =$

Formally Defining δ^*

Definition

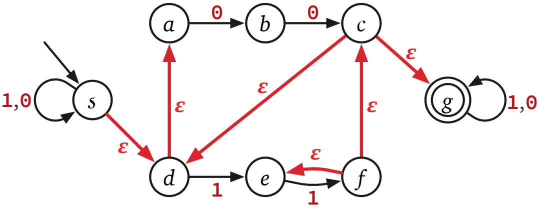
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ-transitions.

Definition

For NFA $N = (Q, \Sigma, \delta, s, A)$, δ^* is a function from $Q \times \Sigma^*$ to $\mathcal{P}(Q)$ defined by

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

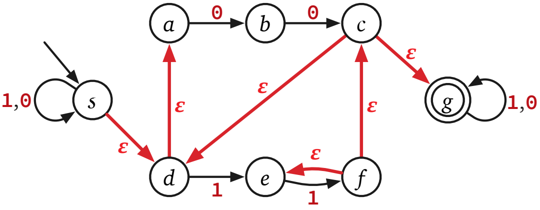
Example



What is:

- $\delta^*(s, \epsilon) =$

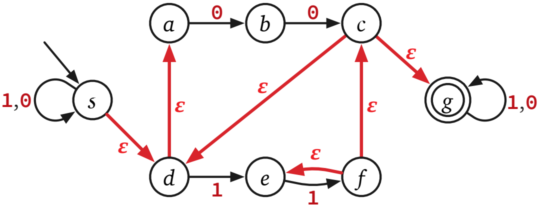
Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) =$

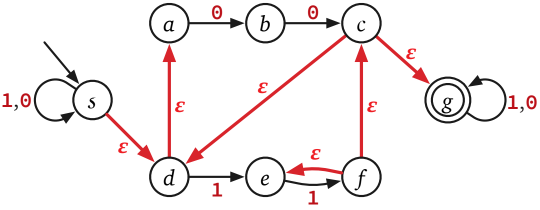
Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 1) =$

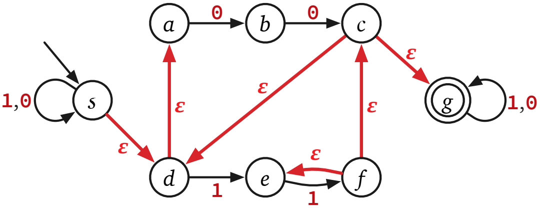
Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 1) = \emptyset$
- $\delta^*(b, 00) =$

Example



What is:

- $\delta^*(s, \epsilon) = \{s, d, a\}$
- $\delta^*(s, 0) = \{s, d, a, b\}$
- $\delta^*(b, 1) = \emptyset$
- $\delta^*(b, 00) = \{b, g\}$

Formally Defining Acceptance

Definition

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

Definition

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

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

Why Are NFAs?

NFAs may seem like a strange model–no "real world" computer behaves non-deterministically!

Why Are NFAs?

NFAs may seem like a strange model–no “real world” computer behaves non-deterministically!

NFAs are very useful as an analysis tool:

- Key middle step in proving the equivalence of DFAs and regular expressions.
- NFAs have more power, so they can be (much) easier to design when you want to show a language is regular.

Why Are NFAs?

NFAs may seem like a strange model–no "real world" computer behaves non-deterministically!

NFAs are very useful as an analysis tool:

- Key middle step in proving the equivalence of DFAs and regular expressions.
- NFAs have more power, so they can be (much) easier to design when you want to show a language is regular.

Non-determinism will come up again in the last portion of the class as a nice characterization of "easily checkable" problems.

Part II

Constructing NFAs

Standard Design Tricks

- Every DFA is a NFA so can just design a DFA

Standard Design Tricks

- Every DFA is a NFA so can just design a DFA
- NFAs provide ability to "guess and verify" which simplifies design and can reduce number of states

Simplification Example

$L = \{$strings that contain 00101 as a substring$\}$

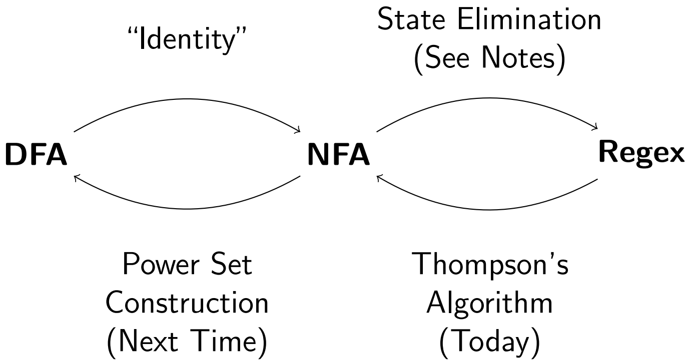
Reduced States Example

$L = \{$strings that have a 1 in the third-to-last position$\}$

Part III

NFAs capture Regular Languages

Roadmap


"Identity"
State Elimination
(See Notes)
DFA
NFA
Regex
Power Set
Construction
(Next Time)
Thompson's
Algorithm
(Today)

Regular Languages Recap

Regular Languages

$\emptyset$ regular
$\{\epsilon\}$ regular
$\{a\}$ regular for $a \in \Sigma$
$R_1 \cup R_2$ regular if both are
$R_1 R_2$ regular if both are
R^* is regular if R is

Regular Expressions

$\emptyset$ denotes $\emptyset$
ϵ denotes $\{\epsilon\}$
$\mathbf{a}$ denote $\{a\}$
$\mathbf{r_1} + \mathbf{r_2}$ denotes $R_1 \cup R_2$
$\mathbf{r_1}\mathbf{r_2}$ denotes $R_1 R_2$
$\mathbf{r}^*$ denote R^*

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language

Some Notation

We'll call an NFA "normal form" if:

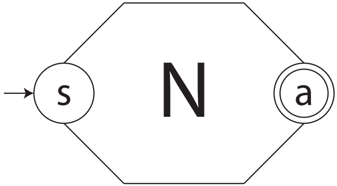
- There is exactly one accepting state, and
- The start state is distinct from the accepting state.

Some Notation

We'll call an NFA "normal form" if:

- There is exactly one accepting state, and
- The start state is distinct from the accepting state.

Pictorially, we can consider an arbitrary NFA N in normal form as:

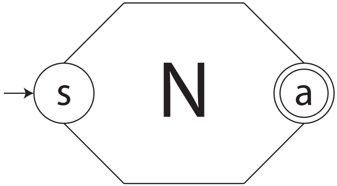


Some Notation

We'll call an NFA "normal form" if:

- There is exactly one accepting state, and
- The start state is distinct from the accepting state.

Pictorially, we can consider an arbitrary NFA N in normal form as:



Exercise for later: given an arbitrary NFA $N = (Q, \Sigma, \delta, s, A)$, how can you modify it to be in normal form?

Thompson's Algorithm: Statement

Theorem

For every regular expression r, *there is a normal form NFA* N *such that* $L(N) = L(r)$.

Thompson's Algorithm: Statement

Theorem

For every regular expression r, *there is a normal form NFA* N *such that* $L(N) = L(r)$.

Proof strategy:

Thompson's Algorithm: Statement

Theorem

For every regular expression r, *there is a normal form NFA* N *such that* $L(N) = L(r)$.

Proof strategy:

- Induction!

Thompson's Algorithm: Statement

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

Proof strategy:

- Induction!
- Given regular expression r, use the NFAs for its constituent parts to construct an NFA for r

Thompson's Algorithm: Statement

Theorem

For every regular expression r*, there is a normal form NFA* N *such that* $L(N) = L(r)$.

Proof strategy:

- Induction!
- Given regular expression r, use the NFAs for its constituent parts to construct an NFA for r
  - Having the smaller NFAs in normal form makes it easier to work with them!

Thompson's Algorithm: Base Cases

Theorem

For every regular expression r*, there is a normal form NFA* N *such that* $L(N) = L(r)$.

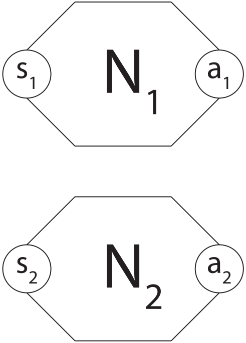
- $r = \emptyset$
- $r = \epsilon$
- $r = a$

Thompson's Algorithm: Inductive Cases

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

- $r = r_1 + r_2$
- $r = r_1 r_2$
- $r = r_1^*$

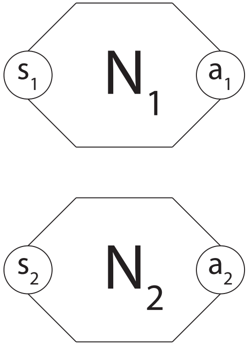


Thompson's Algorithm: Union

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1 + r_2$

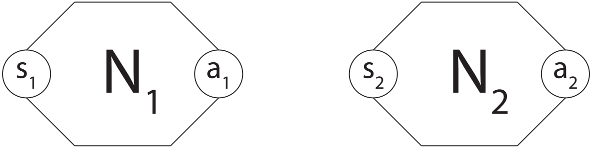


Thompson's Algorithm: Concatenation

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1 r_2$

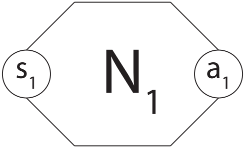


Thompson's Algorithm: Kleene Star

Theorem

For every regular expression r, *there is a normal form NFA* N *such that* $L(N) = L(r)$.

$r = r_1^*$

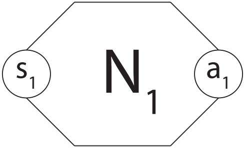


Thompson's Algorithm: Kleene Star (Attempt 2)

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1^*$

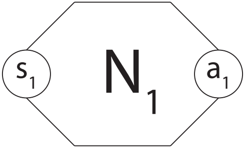


Thompson's Algorithm: Kleene Star (Attempt 3)

Theorem

For every regular expression r, there is a normal form NFA N such that $L(N) = L(r)$.

$r = r_1^*$



Thompson's Algorithm: Example

$r = 0(0+1)^*$