Non-deterministic Finite Automata (NFAs)

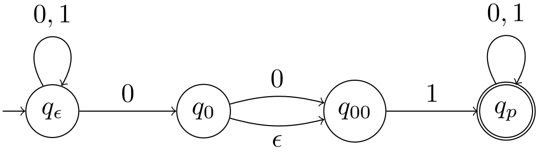
Lecture 4

January 26, 2023

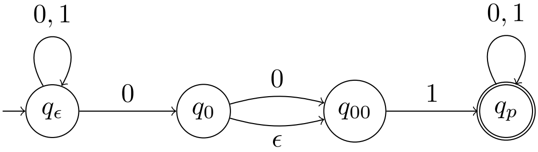
Part I

NFA Introduction

Non-deterministic Finite State Automata (NFAs)



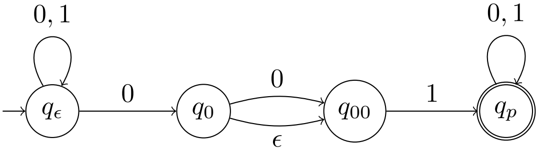
Non-deterministic Finite State Automata (NFAs)



Differences from DFA

- From state q on same letter $a \in \Sigma$ multiple possible states
- No transitions from q on some letters
- ϵ-transitions!

Non-deterministic Finite State Automata (NFAs)



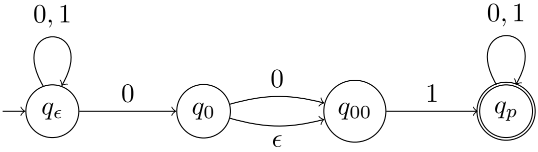
Differences from DFA

- From state q on same letter $a \in \Sigma$ multiple possible states
- No transitions from q on some letters
- ϵ-transitions!

Questions:

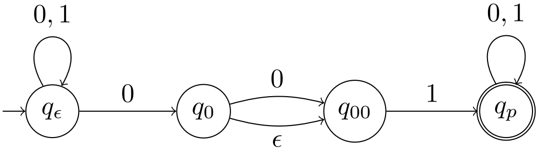
- Is this a "real" machine?
- What does it do?

NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

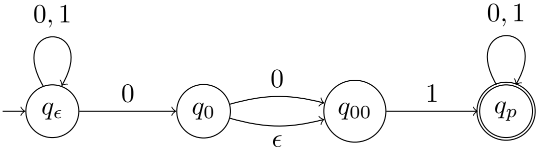
NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

- From q_ϵ on 1

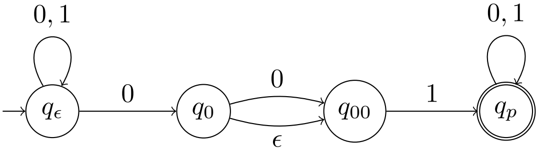
NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

- From q_ϵ on 1
- From q_ϵ on 0

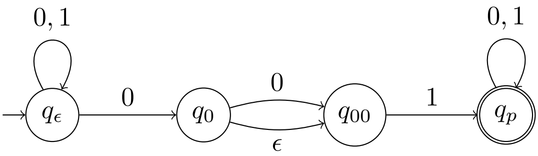
NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

- From q_ϵ on 1
- From q_ϵ on 0
- From q_0 on ϵ

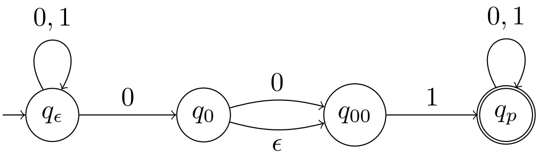
NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

- From q_ϵ on 1
- From q_ϵ on 0
- From q_0 on ϵ
- From q_ϵ on 01

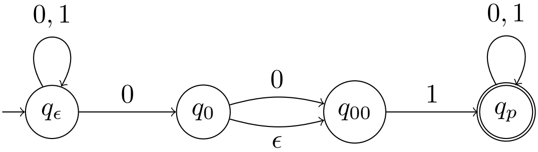
NFA behavior



Machine on input string w from state q can lead to *set of states* (could be empty)

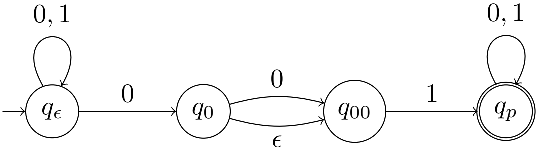
- From q_ϵ on 1
- From q_ϵ on 0
- From q_0 on ϵ
- From q_ϵ on 01
- From q_{00} on 00

NFA acceptance: informal



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

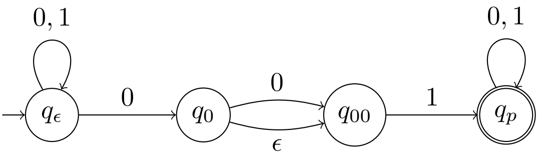
NFA acceptance: informal



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

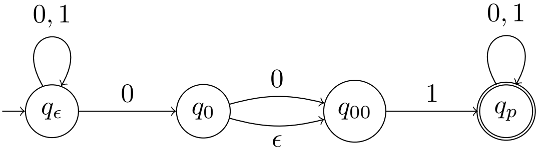
The language accepted (or recognized) by a NFA N is denote by $L(N)$ and defined as: $L(N) = \{w \mid N \text{ accepts } w\}$.

NFA acceptance: example



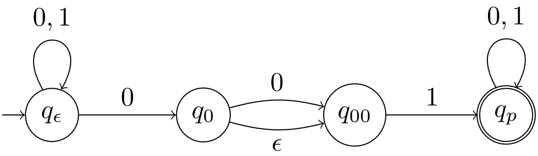
- Is 01 accepted?

NFA acceptance: example



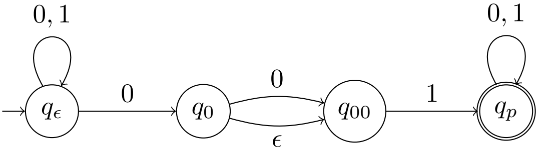
- Is 01 accepted?
- Is 001 accepted?

NFA acceptance: example



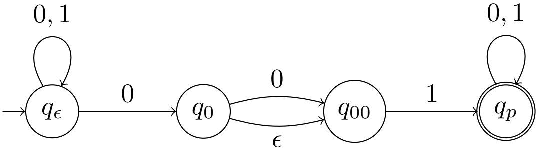
- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?

NFA acceptance: example



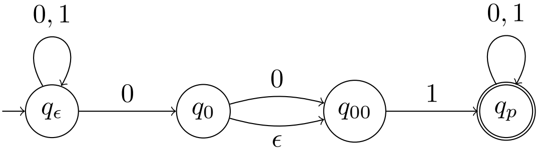
- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 accepted?

NFA acceptance: example



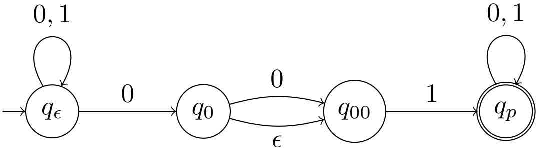
- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 accepted?
- What is the language accepted by N?

NFA acceptance: example



- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 accepted?
- What is the language accepted by ***N***?

NFA acceptance: example



- Is 01 accepted?
- Is 001 accepted?
- Is 100 accepted?
- Are all strings in 1*01 accepted?
- What is the language accepted by ***N***?

Comment: Unlike DFAs, it is easier in NFAs to show that a string is accepted than to show that a string is **not** accepted.

Formal Tuple Notation

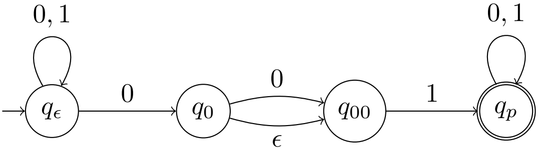
Definition

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.

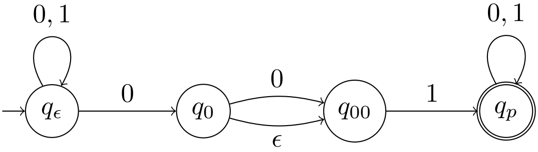
$\delta(q, a)$ for $a \in \Sigma \cup \{\epsilon\}$ is a susbet of Q — a set of states.

Example



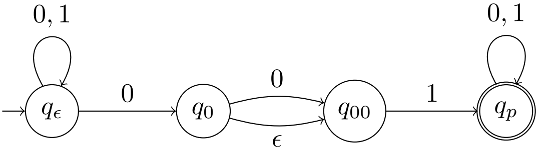
- $Q =$

Example



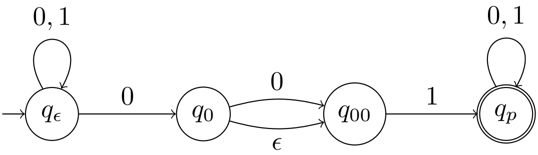
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$

Example



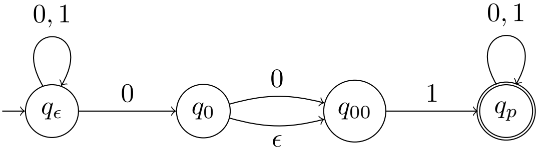
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma =$

Example



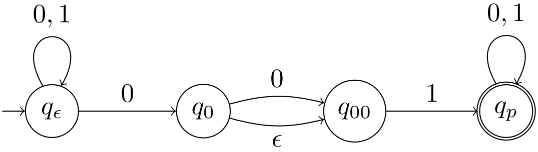
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$

Example



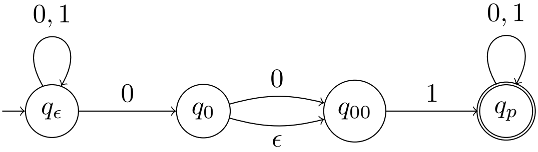
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- δ

Example



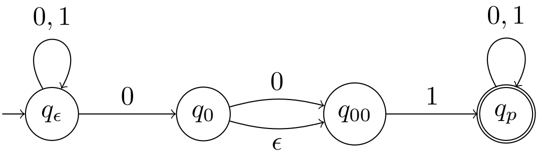
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- δ
- $s =$

Example



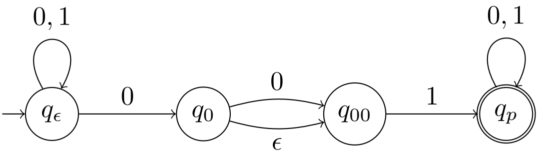
- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_\epsilon$

Example



- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_\epsilon$
- $A =$

Example



- $Q = \{q_\epsilon, q_0, q_{00}, q_p\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = q_\epsilon$
- $A = \{q_p\}$

Extending the transition function to strings

Given NFA $N = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is a *set of states* that N can go to from q on reading $a \in \Sigma \cup \{\epsilon\}$.

Extending the transition function to strings

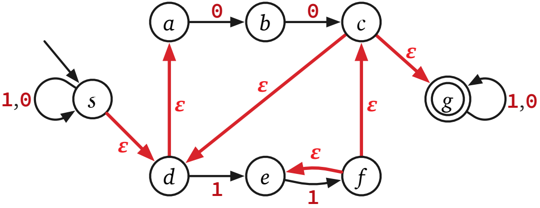
Given NFA $N = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is a *set of states* that N can go to from q on reading $a \in \Sigma \cup \{\epsilon\}$.

Want transition function $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ where $\delta^*(q, w)$ is the set of states that can be reached by N on input w starting in state q.

Extending the transition function to strings

Definition

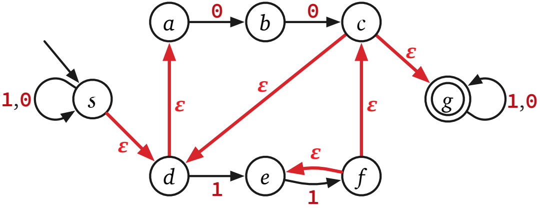
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon\text{reach}(q)$ is the set of all states that q can reach using only ϵ-transitions.



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.



- $\epsilon\text{reach}(s) =$
- $\epsilon\text{reach}(b) =$
- $\epsilon\text{reach}(f) =$

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 $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$

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 $a \in \Sigma$
  $\delta^*(q, a) = \cup_{p \in \epsilon\text{reach}(q)}(\cup_{r \in \delta(p,a)} \epsilon\text{reach}(r))$

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 $w = ax$,
  $\delta^*(q, w) = \cup_{p \in \delta^*(q,a)} \delta^*(p, x)$

Formal definition of language accepted by N

Definition

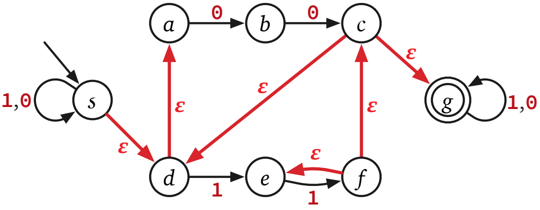
A string w is accepted by NFA N if $\delta^*_N(s, w) \cap A \neq \emptyset$.

Definition

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

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

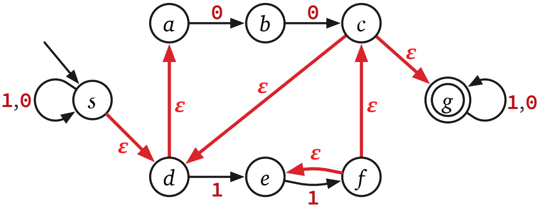
Example



What is:

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

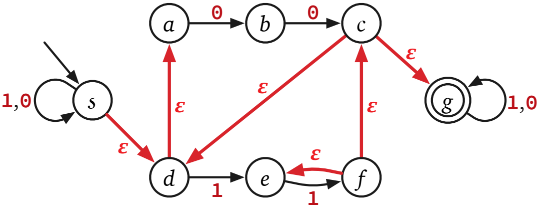
Example



What is:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$

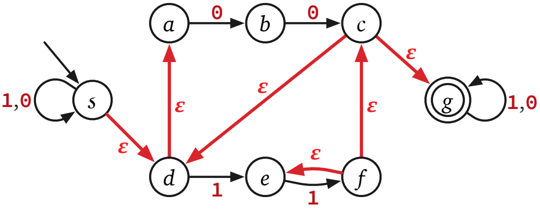
Example



What is:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$

Example



What is:

- $\delta^*(s, \epsilon)$
- $\delta^*(s, 0)$
- $\delta^*(c, 0)$
- $\delta^*(b, 00)$

Another definition of computation

Definition

A state p is reachable from q on w denoted by $q \xrightarrow{w}_N p$ if there exists a sequence of states $r_0, r_1, \ldots, r_k$ and a sequence $x_1, x_2, \ldots, x_k$ where $x_i \in \Sigma \cup \{\epsilon\}$ for each i, such that:

- $r_0 = q$,
- for each i, $r_{i+1} \in \delta(r_i, x_{i+1})$,
- $r_k = p$, and
- $w = x_1 x_2 x_3 \cdots x_k$.

Definition

$\delta^* N(q, w) = \{p \in Q \mid q \xrightarrow{w}_N p\}$.

Why non-determinism?

- Non-determinism adds power to the model; richer programming language and hence (much) easier to "design" programs
- Fundamental in **theory** to prove many theorems
- Very important in **practice** directly and indirectly
- Many deep connections to various fields in Computer Science and Mathematics
- Michael Rabin and Dana Scott introduced non-determinism in a 1959 paper titled "Finite Automata and Their Decision Problems". They won the Turing Award in 1976 partly for this contribution.

Several interpretations of non-determinism. Hard to understand at the outset. Over time you will get used to it and appreciate the conceptual value.

Part II

Constructing NFAs

DFAs and NFAs

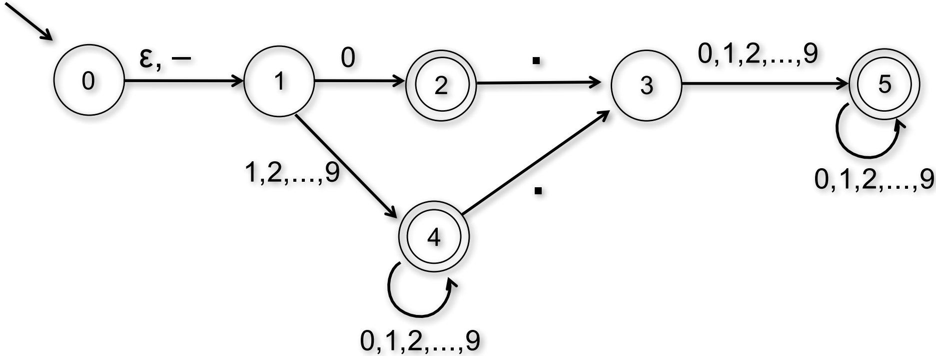
- Every DFA is a NFA so NFAs are at least as powerful as DFAs.
- NFAs prove ability to "guess and verify" which simplifies design and reduces number of states
- Easy proofs of some closure properties

Example

Strings that represent decimal numbers.

Example

Strings that represent decimal numbers.



Example

- {strings that contain CS374 as a substring}

Example

- {strings that contain CS374 as a substring}
- {strings that contain CS374 or CS473 as a substring}

Example

- {strings that contain CS374 as a substring}
- {strings that contain CS374 or CS473 as a substring}
- {strings that contain CS374 and CS473 as substrings}

Example

$L_k = \{$bitstrings that have a 1 k positions from the end$\}$

A simple transformation

Theorem

For every NFA N there is another NFA N' such that $L(N) = L(N')$ and such that N' has the following two properties:

- *N' has single final state f that has no outgoing transitions*
- *The start state s of N is different from f*

Part III

Closure Properties of NFAs

Closure properties of NFAs

Are the class of languages accepted by NFAs closed under the following operations?

- union
- intersection
- concatenation
- Kleene star
- complement

Closure under union

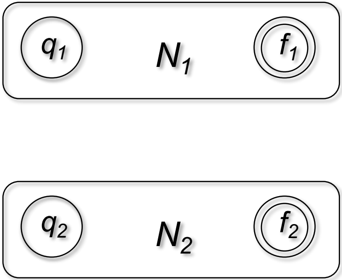
Theorem

For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cup L(N_2)$.

Closure under union

Theorem

For any two NFAs N_1 *and* N_2 *there is a NFA* N *such that* $L(N) = L(N_1) \cup L(N_2)$.



Closure under concatenation

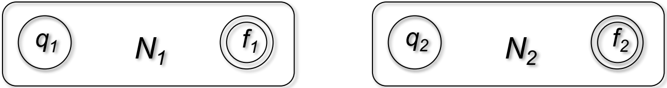
Theorem

For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \bullet L(N_2)$.

Closure under concatenation

Theorem

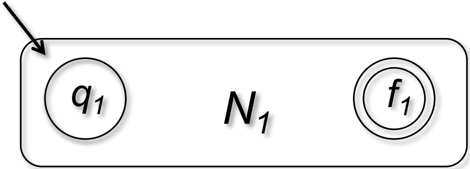
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \bullet L(N_2)$.



Closure under Kleene star

Theorem

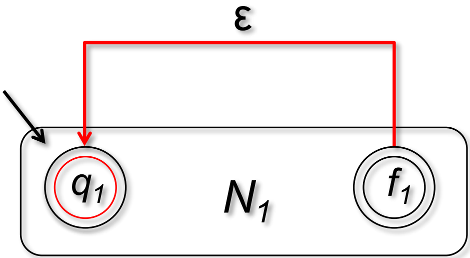
For any NFA N_1 *there is a NFA* N *such that* $L(N) = (L(N_1))^*$.



Closure under Kleene star

Theorem

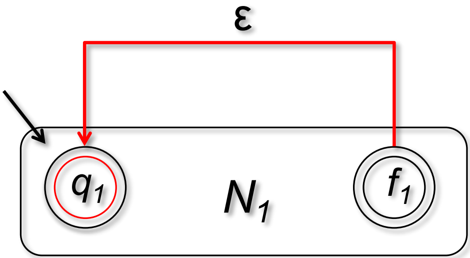
For any NFA N_1 *there is a NFA* N *such that* $L(N) = (L(N_1))^*$.



Closure under Kleene star

Theorem

For any NFA N_1 *there is a NFA* N *such that* $L(N) = (L(N_1))^*$.

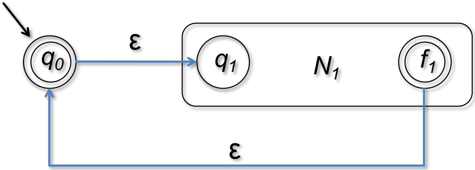


Does not work! Why?

Closure under Kleene star

Theorem

For any NFA $\mathbf{N}_1$ *there is a NFA* $\mathbf{N}$ *such that* $\mathbf{L}(\mathbf{N}) = (\mathbf{L}(\mathbf{N}_1))^*$.



Part IV

NFAs capture Regular Languages

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 $\{\epsilon\}$
a denote $\{a\}$
$\mathrm{r}_1 + \mathrm{r}_2$ denotes $R_1 \cup R_2$
$\mathrm{r}_1\mathrm{r}_2$ denotes $R_1 R_2$
r^* denote R^*

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

NFAs and Regular Language

Theorem

For every regular language L there is an NFA N such that $L = L(N)$.

Proof strategy:

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Base cases: $\emptyset$, $\{\epsilon\}$, $\{a\}$ for $a \in \Sigma$

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$.

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA N s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA N s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$
- $r = r_1 \bullet r_2$.

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA N s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$
- $r = r_1 \bullet r_2$. Use closure of NFA languages under concatenation

NFAs and Regular Language

- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA N s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$
- $r = r_1 \bullet r_2$. Use closure of NFA languages under concatenation
- $r = (r_1)^*$.

NFAs and Regular Language

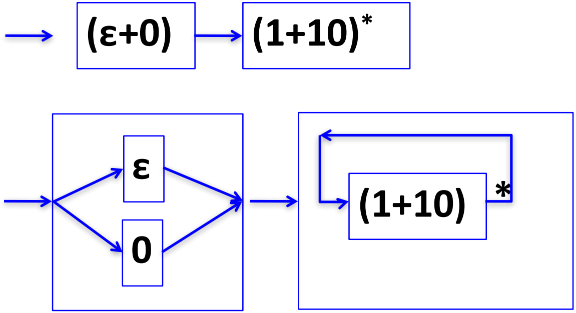
- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

Inductive cases:

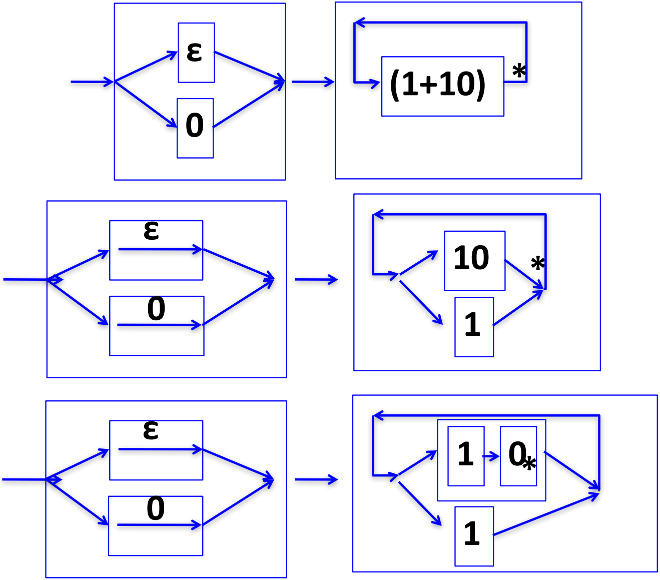
- r_1, r_2 regular expressions and $r = r_1 + r_2$.
  By induction there are NFAs N_1, N_2 s.t
  $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We have already seen that there is NFA N s.t $L(N) = L(N_1) \cup L(N_2)$, hence $L(N) = L(r)$
- $r = r_1 \bullet r_2$. Use closure of NFA languages under concatenation
- $r = (r_1)^*$. Use closure of NFA languages under Kleene star

Example

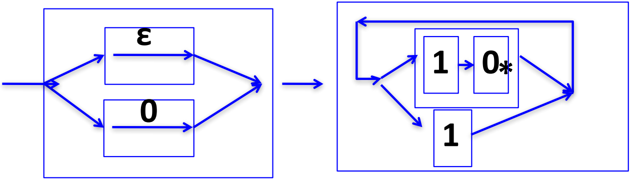
$(\varepsilon+0)(1+10)^*$

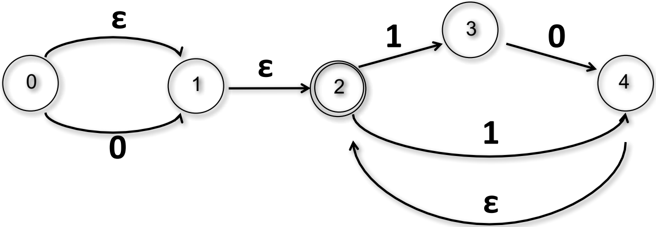


Example



Example





Final NFA simplified slightly to reduce states

Summary and Skills

NFAs introduce and showcase the power of *non-determinism* in computation. It is a mathematical construct. Important to digest the formal definitions.

- How does the definition of an NFA differ from that of a DFA?
- What is the definiton of $L(N)$ where N is an NFA?
- In particular, what is the formal definition of the transition function δ and the transition function δ^*?
- What is an algorithm to check whether a string w is accepted by an NFA N?
- What is an algorithm to chek whether a string w is *not* accepted by an NFA N?

NFAs generalize DFAs. Why do NFAs generalize regular expressions? Thompson's algorithm to convert a regular expression to an NFA.

Summary and Skills

Designing NFAs for a language

- NFAs generalize DFAs
- Languages accepted by NFAs are closed under union, *concatenation*, and Kleene star. Hence NFAs also generalize regular expressions/languages.
- Non-determinism allows for "guess" and "verify" approach for computation and hence one can design simpler machines. Understanding this approach is necessary to use NFAs to show many non-trivial closure properties of regular languages.
- Can combining the power of the three preceding properties