Non-deterministic Finite Automata (NFAs)

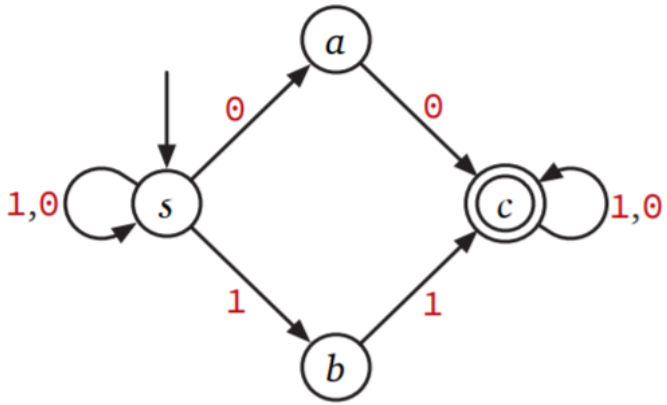
Lecture 5

Feb 2, 2020

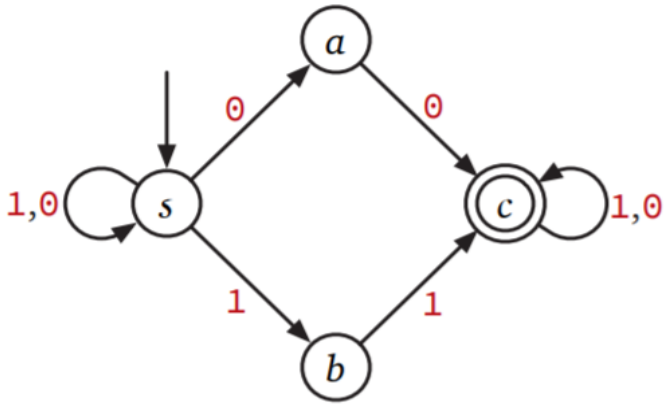
Part I

NFA Introduction

Non-deterministic Finite State Automata (NFAs)



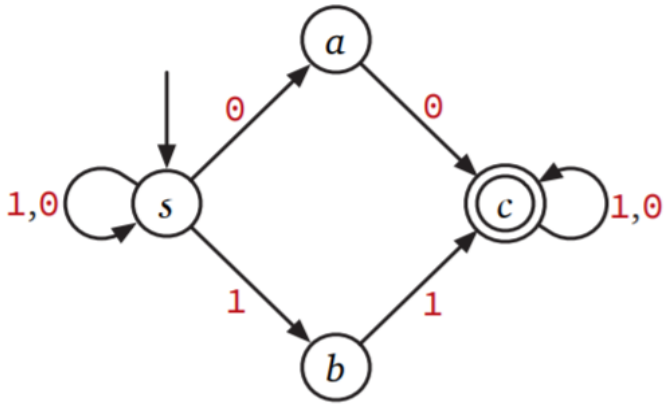
Non-deterministic Finite State Automata (NFAs)



Differences from DFA

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

Non-deterministic Finite State Automata (NFAs)



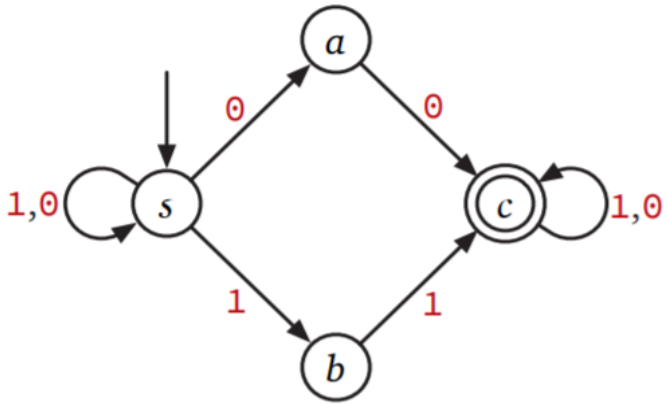
Differences from DFA

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

Questions:

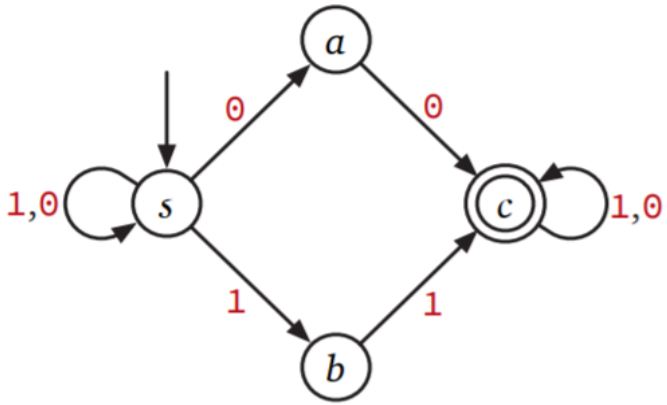
- Is this a "real" machine?
- What does it do?

NFA behavior



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

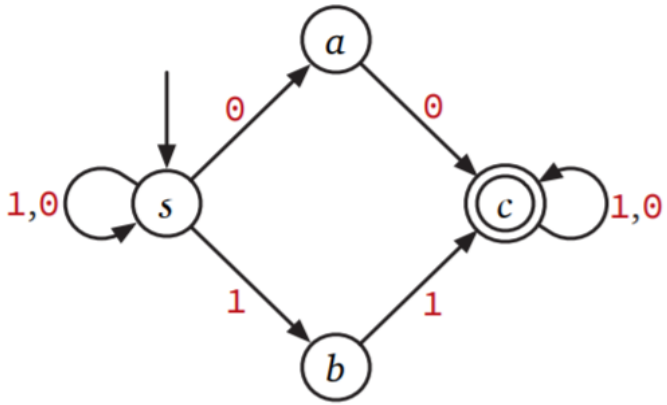
NFA behavior



Machine on input symbol a from state q can lead to set of states (could be empty)

- From s on $\mathbf{0}$

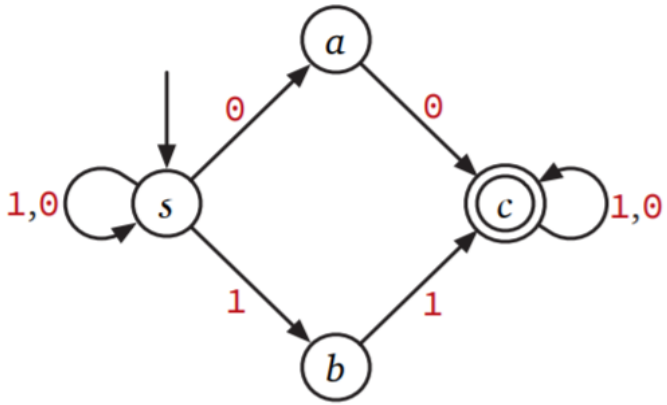
NFA behavior



Machine on input symbol a from state q can lead to set of states (could be empty)

- From s on **0**
- From a on **0**

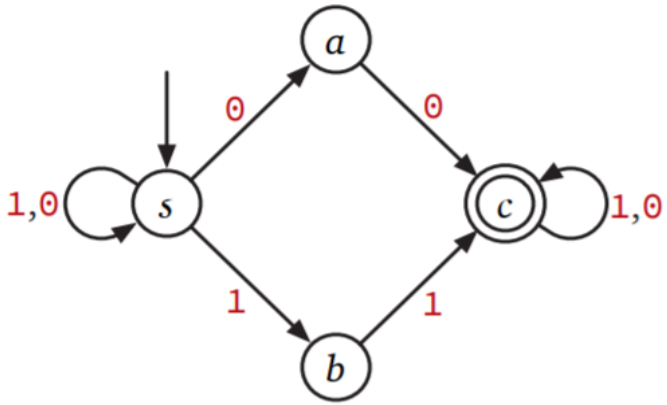
NFA behavior



Machine on input symbol a from state q can lead to set of states (could be empty)

- From s on **0**
- From a on **0**
- From s on **1**

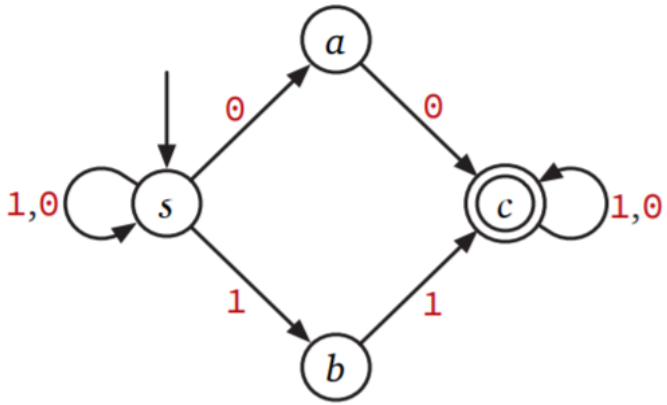
NFA behavior



Machine on input symbol a from state q can lead to set of states (could be empty)

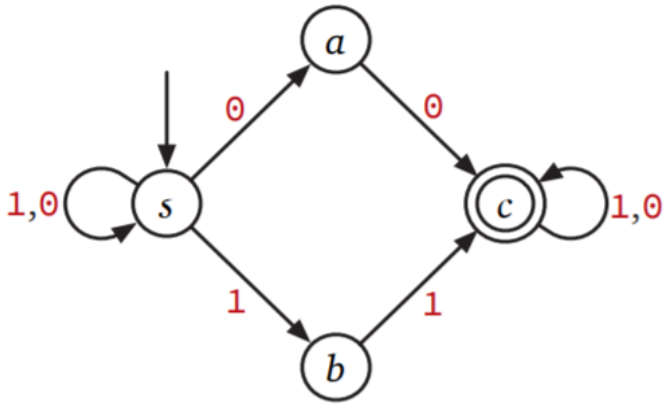
- From s on **0**
- From a on **0**
- From s on **1**
- From a on **1**

NFA behavior



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

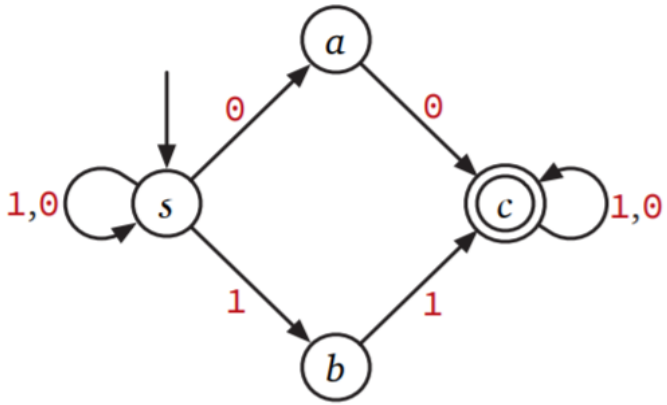
NFA behavior



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

- From s on 11

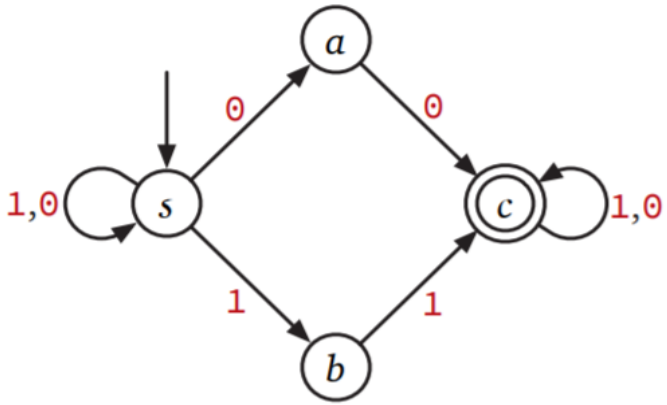
NFA behavior



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

- From s on **11**
- From s on **10**

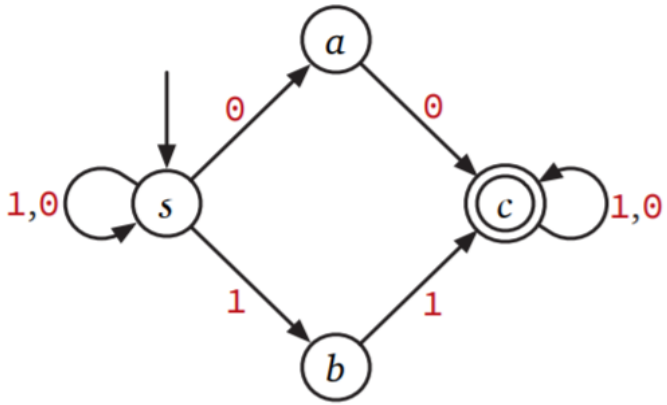
NFA behavior



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

- From s on **11**
- From s on **10**
- From b on **01**

NFA behavior



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

- From s on **11**
- From s on **10**
- From b on **01**
- From b on **11**

Formal Tuple Notation

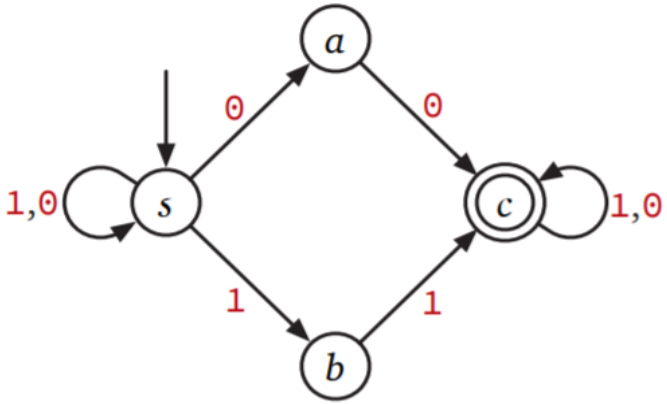
Definition

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,
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ 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.

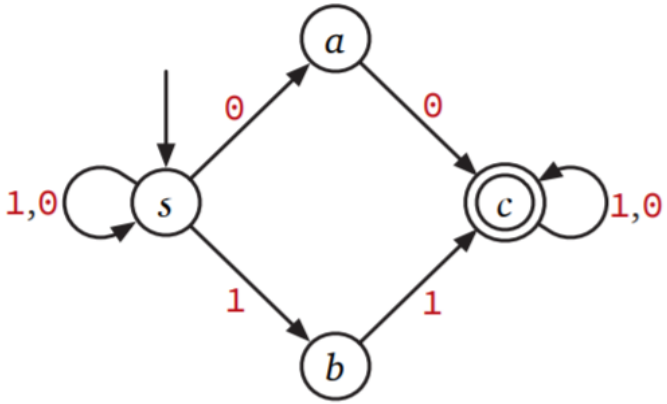
$\delta(q, a)$ for $a \in \Sigma$ is a susbet of Q — a set of states.

Example



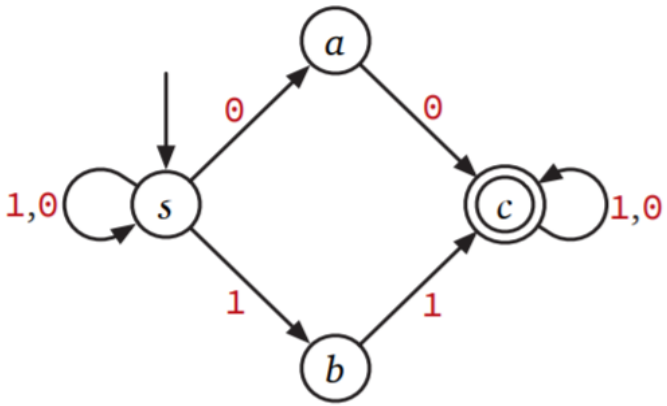
- $Q =$

Example



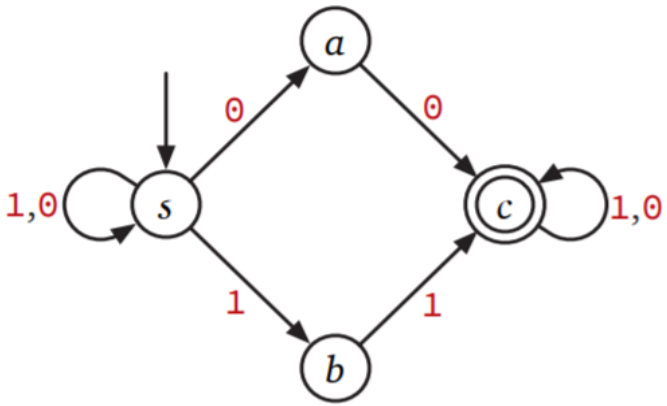
- $Q = \{s, a, b, c\}$

Example



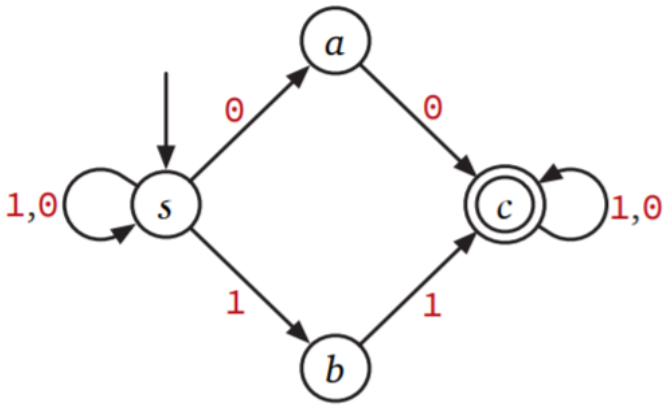
- $Q = \{s, a, b, c\}$
- $\Sigma =$

Example



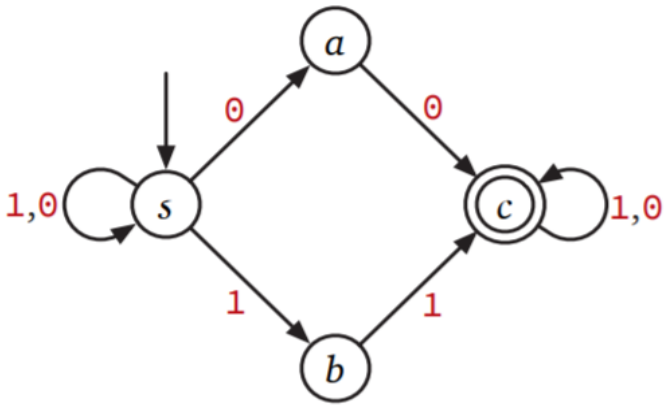
- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$

Example



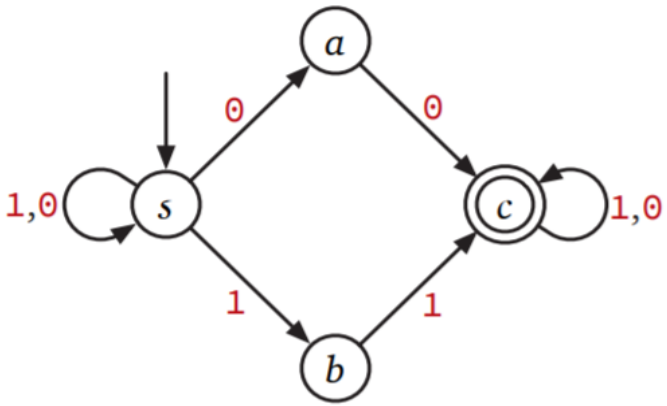
- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$
- δ

Example



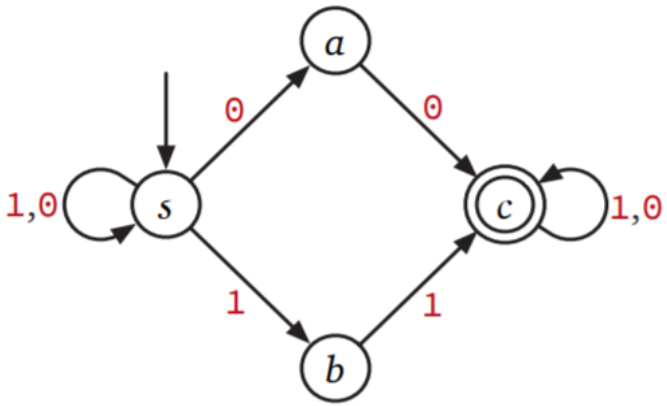
- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$
- δ
- $s =$

Example



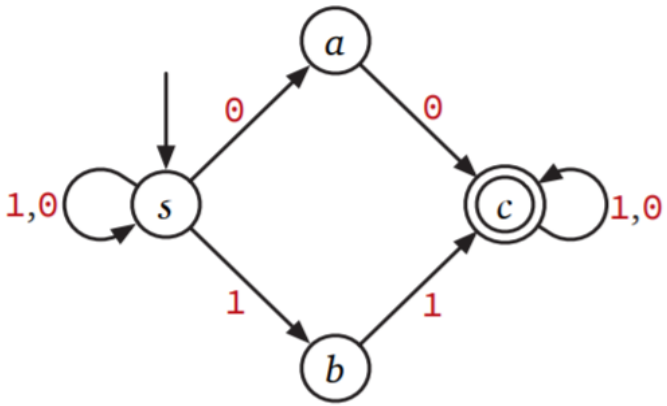
- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = s$

Example



- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = s$
- $A =$

Example



- $Q = \{s, a, b, c\}$
- $\Sigma = \{0, 1\}$
- δ
- $s = s$
- $A = \{c\}$

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.

$$\delta^*(q, w) = \begin{cases} \{q\} & \text{if } w = \epsilon \\ \cup_{r \in \delta(q,a)} \delta^*(r, x) & \text{if } w = ax \end{cases}$$

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

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

Formal definition of language accepted by N

Definition

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

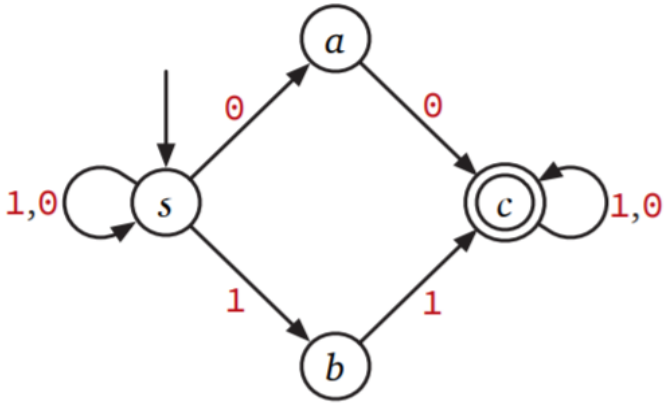
Definition

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

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

NFAs subsumes DFAs!

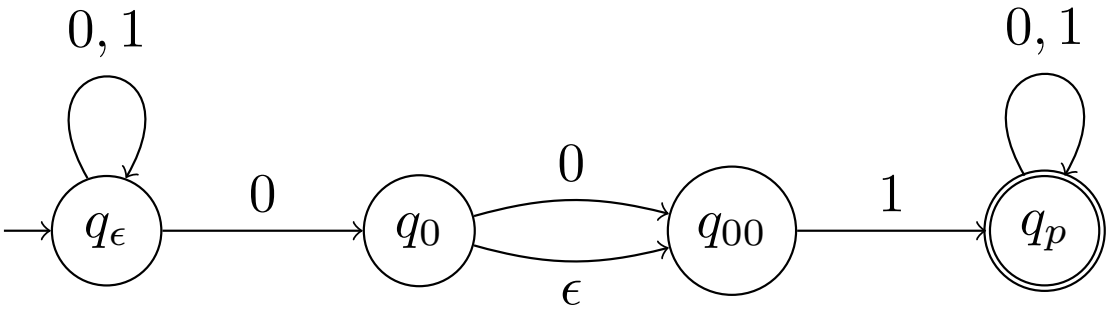
Example



Part II

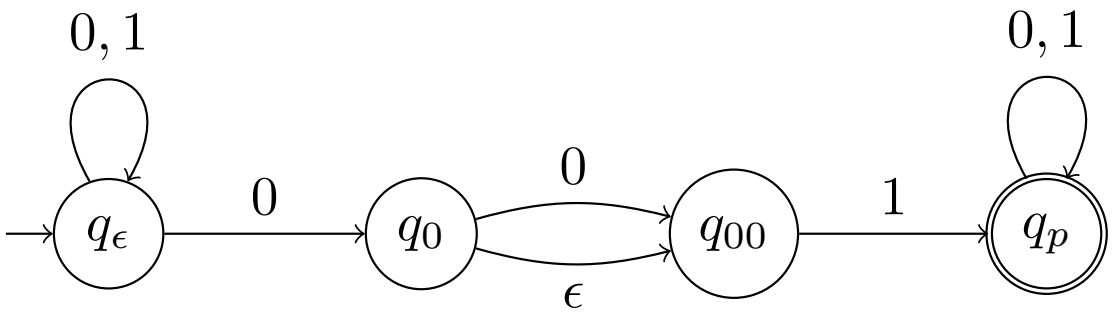
NFA with ϵ Transitions

NFA behavior



Can move from q_0 to q_{00} without using any symbol.

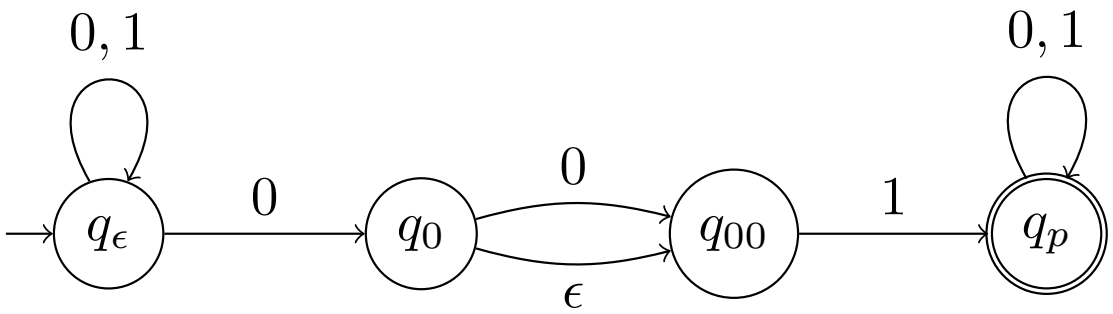
NFA behavior



Can move from q_0 to q_{00} without using any symbol.

- From q_ϵ on **1**

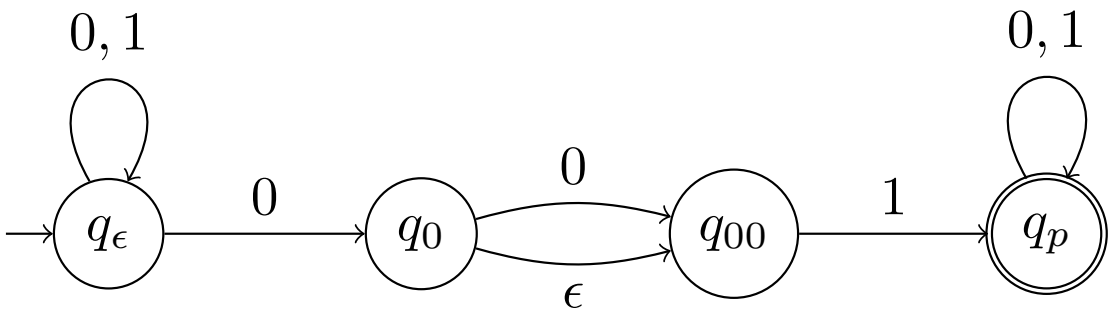
NFA behavior



Can move from q_0 to q_{00} without using any symbol.

- From q_ϵ on **1**
- From q_ϵ on **0**

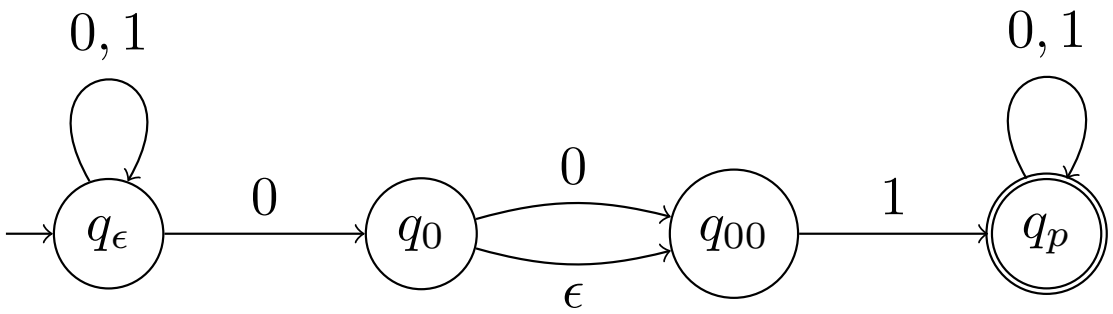
NFA behavior



Can move from q_0 to q_{00} without using any symbol.

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

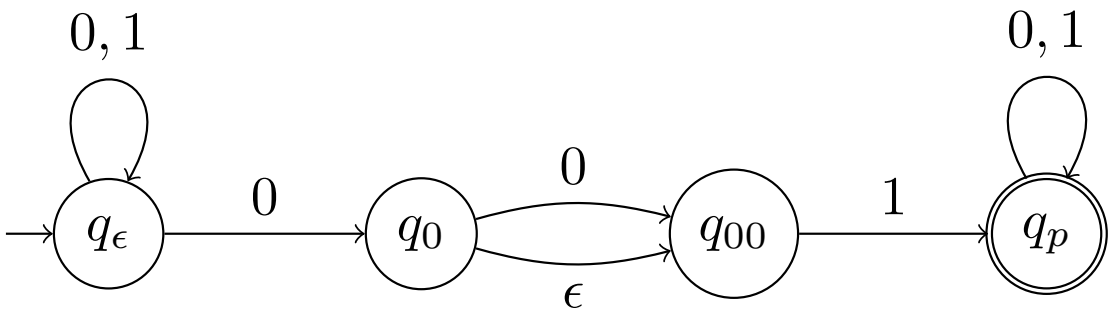
NFA behavior



Can move from q_0 to q_{00} without using any symbol.

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

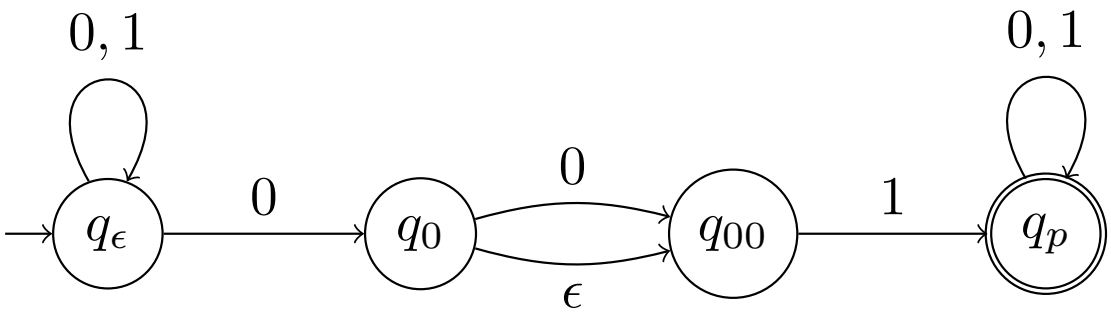
NFA behavior



Can move from q_0 to q_{00} without using any symbol.

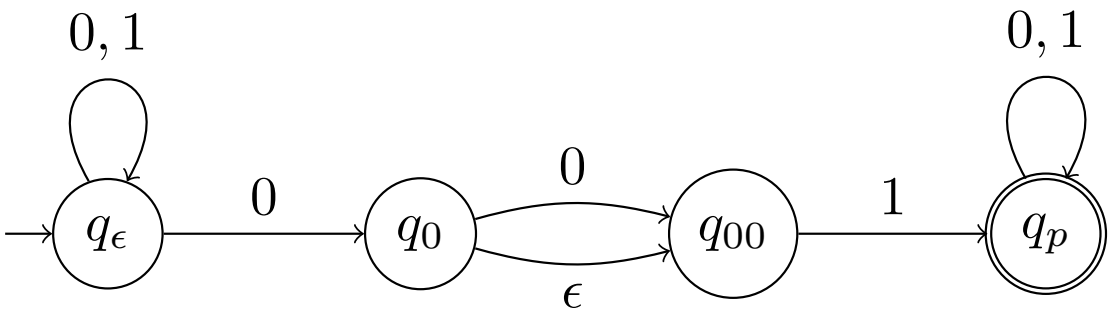
- From q_ϵ on **1**
- From q_ϵ on **0**
- From q_0 on ϵ
- From q_ϵ on **01**
- From q_{00} on **00**

NFA acceptance: example



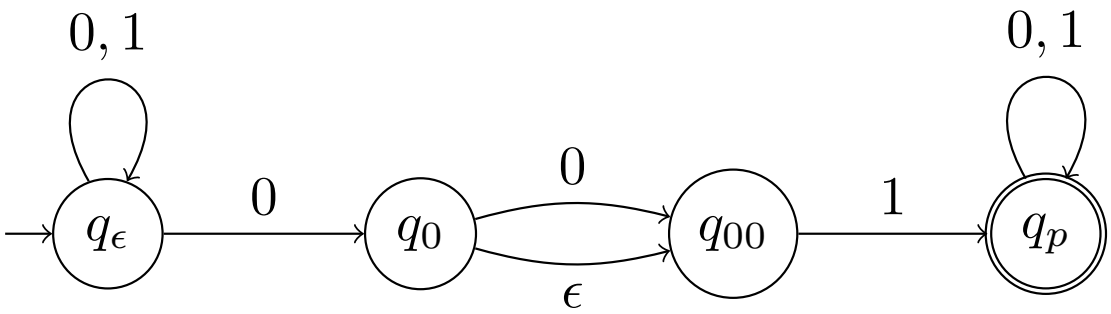
- Is **01** accepted?

NFA acceptance: example



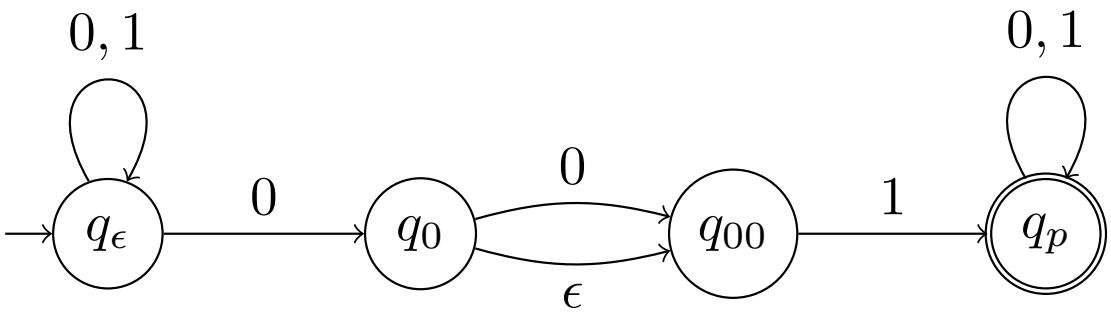
- Is **01** accepted?
- Is **001** accepted?

NFA acceptance: example



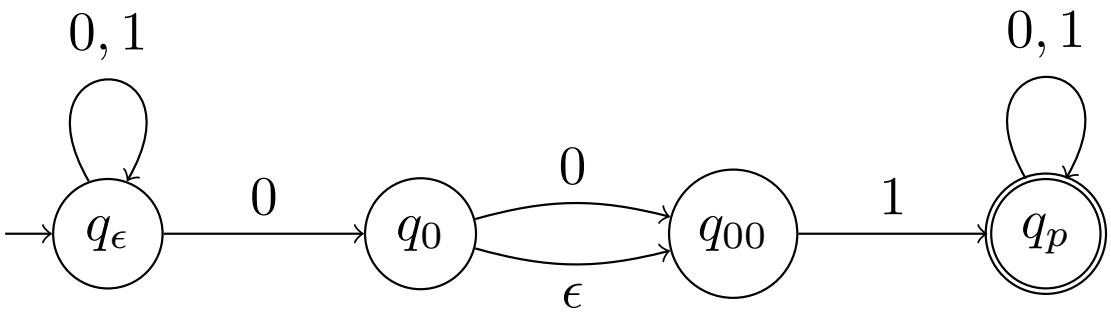
- Is **01** accepted?
- Is **001** accepted?
- Is **100** accepted?

NFA acceptance: example



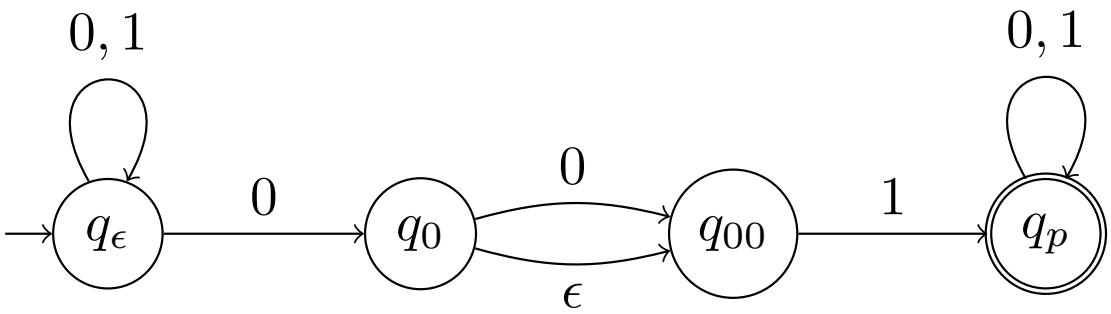
- Is **01** accepted?
- Is **001** accepted?
- Is **100** accepted?
- Are all strings in **1*01** accepted?

NFA acceptance: example



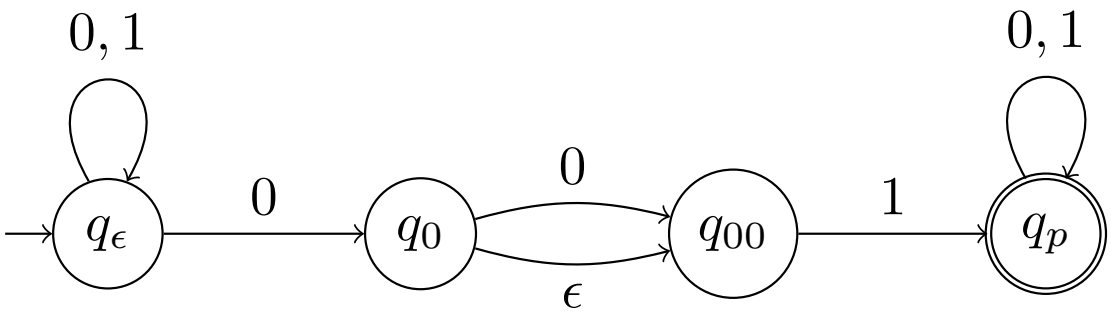
- Is **01** accepted?
- Is **001** accepted?
- Is **100** accepted?
- Are all strings in **1*01** accepted?
- What is the language accepted by ***N***?

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

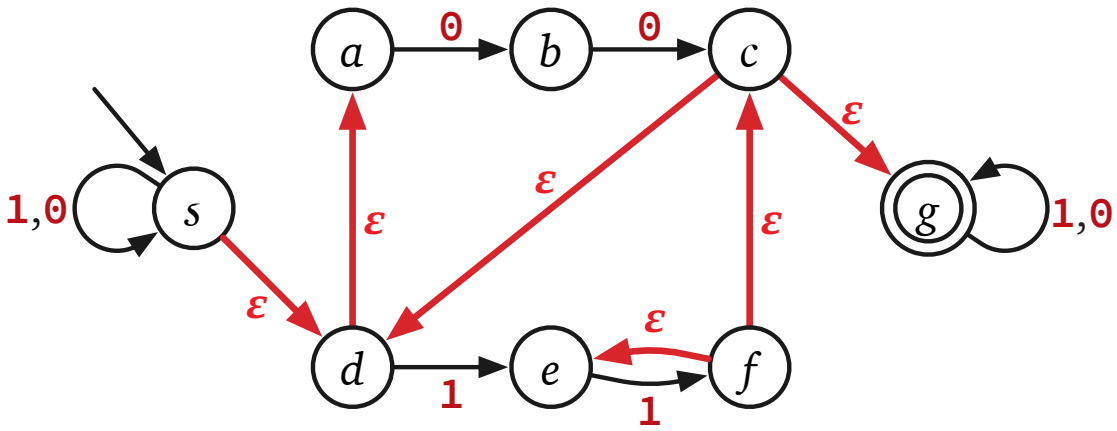
- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q)$ is the transition function (here $\mathcal{P}(Q)$ is the power set of Q)
- $s \in Q$ is the start state,
- $A \subseteq Q$ is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

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

Extending the transition function to strings

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.



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.

Abuse notation: $\epsilon\text{reach}(S) = \cup_{p \in S} \epsilon\text{reach}(p)$,

Extending the transition function to strings

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.

Abuse notation: $\epsilon\text{reach}(S) = \cup_{p \in S} \epsilon\text{reach}(p)$,

$\delta(S, a) = \cup_{p \in S} \delta(p, a)$, $\quad \delta^*(S, w) = \cup_{p \in S} \delta^*(p, w)$

Extending the transition function to strings

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.

Abuse notation: $\epsilon\text{reach}(S) = \cup_{p \in S} \epsilon\text{reach}(p)$,
$\delta(S, a) = \cup_{p \in S} \delta(p, a)$, $\delta^*(S, w) = \cup_{p \in S} \delta^*(p, w)$

Definition

Inductive definition of $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

- 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.

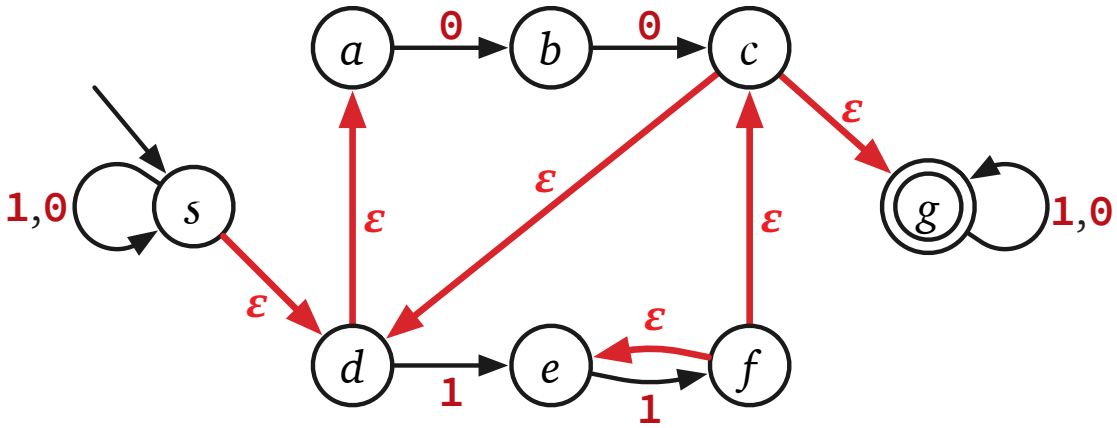
Abuse notation: $\epsilon\text{reach}(S) = \cup_{p \in S} \epsilon\text{reach}(p)$,
$\delta(S, a) = \cup_{p \in S} \delta(p, a)$, $\delta^*(S, w) = \cup_{p \in S} \delta^*(p, w)$

Definition

Inductive definition of $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon\text{reach}(q)$
- if $w = ax$, $\delta^*(q, w) = \delta^*(\delta(\epsilon\text{reach}(q), a), x)$

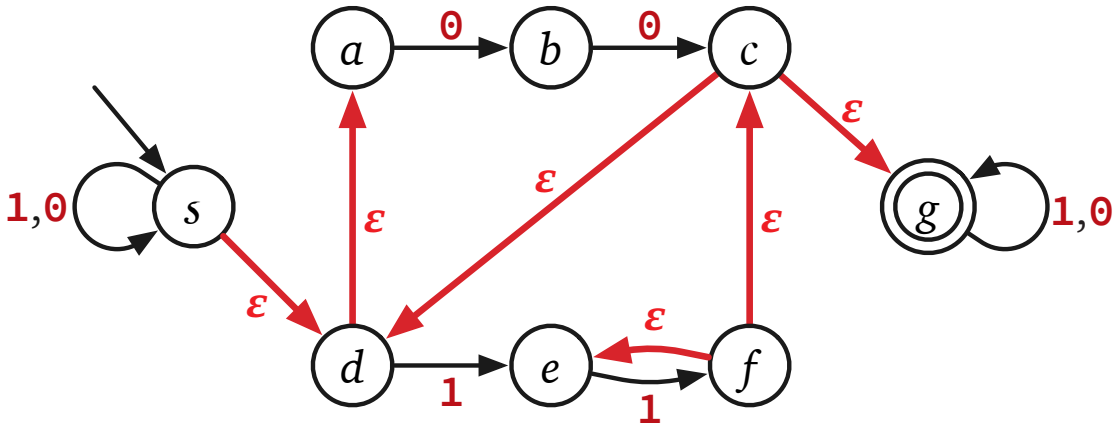
Example



What is:

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

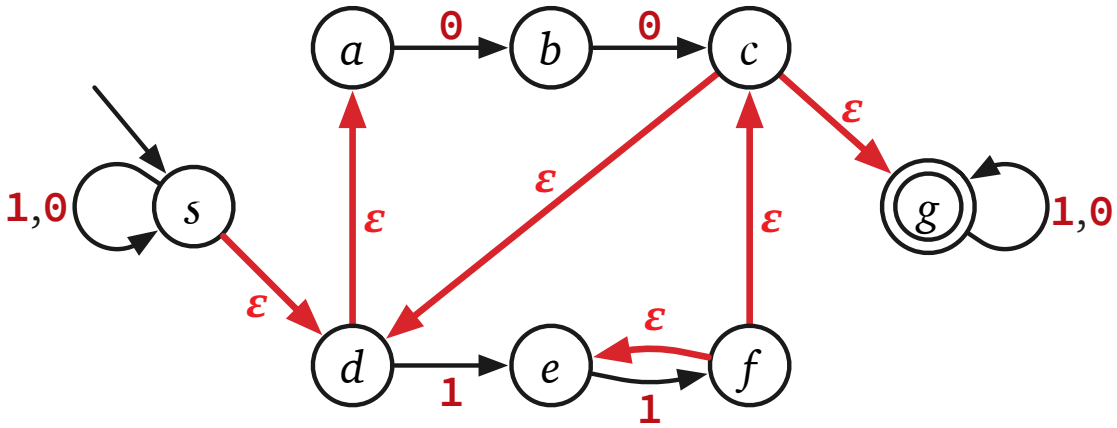
Example



What is:

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

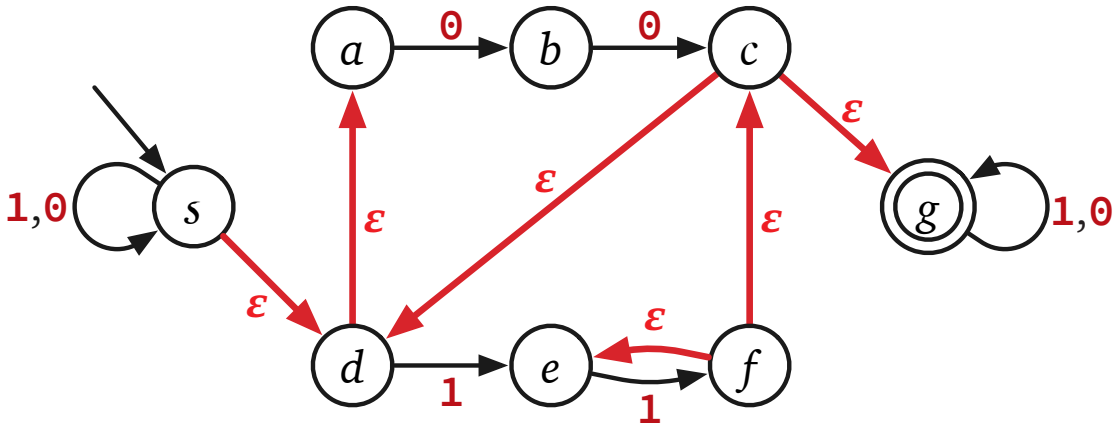
Example



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

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

Equivalence of NFA w/o ϵ-Transitions

Lemma

Given an NFA $M = (\Sigma, Q, s, A, \delta)$ with ϵ-transitions there is an equivalent NFA $N = (\Sigma, Q', s', A', \delta')$ without them.

Equivalence of NFA w/o ϵ-Transitions

Lemma

Given an NFA $M = (\Sigma, Q, s, A, \delta)$ with ϵ-transitions there is an equivalent NFA $N = (\Sigma, Q', s', A', \delta')$ without them.

$$
\begin{aligned}
Q' &= Q \\
s' &= s \\
A' &=
\end{aligned}
$$

Equivalence of NFA w/o ϵ-Transitions

Lemma

Given an NFA $M = (\Sigma, Q, s, A, \delta)$ with ϵ-transitions there is an equivalent NFA $N = (\Sigma, Q', s', A', \delta')$ without them.

$$
\begin{aligned}
Q' &= Q \\
s' &= s \\
A' &= \{q \in Q \mid \epsilon\text{reach}(q) \cap A \neq \emptyset\} \\
\delta'(q, a) &=
\end{aligned}
$$

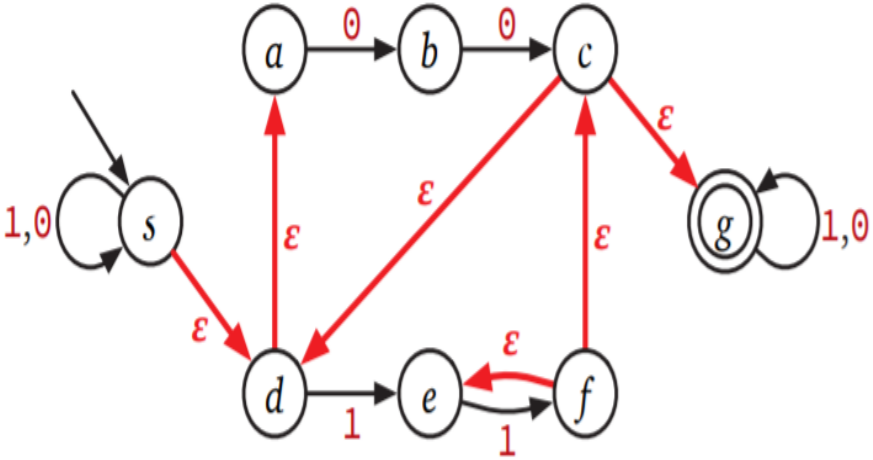
Equivalence of NFA w/o ϵ-Transitions

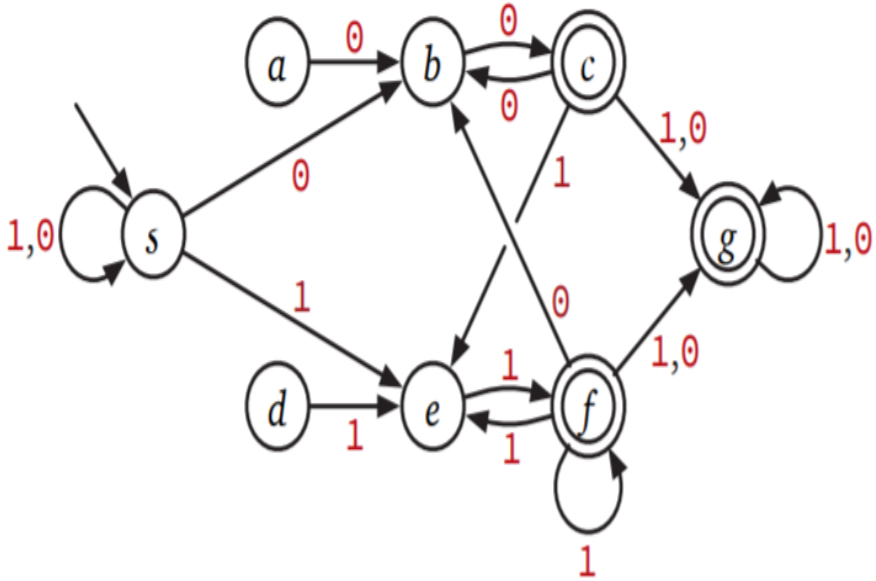
Lemma

Given an NFA $M = (\Sigma, Q, s, A, \delta)$ with ϵ-transitions there is an equivalent NFA $N = (\Sigma, Q', s', A', \delta')$ without them.

$$
\begin{aligned}
Q' &= Q \\
s' &= s \\
A' &= \{q \in Q \mid \epsilon\text{reach}(q) \cap A \neq \emptyset\} \\
\delta'(q, a) &= \delta(\epsilon\text{reach}(q), a)
\end{aligned}
$$

Example





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

Many interpretations of non-determinism. Hard to understand at the outset. Get used to it and then you will appreciate it slowly.

Part III

Constructing NFAs

DFAs and NFAs

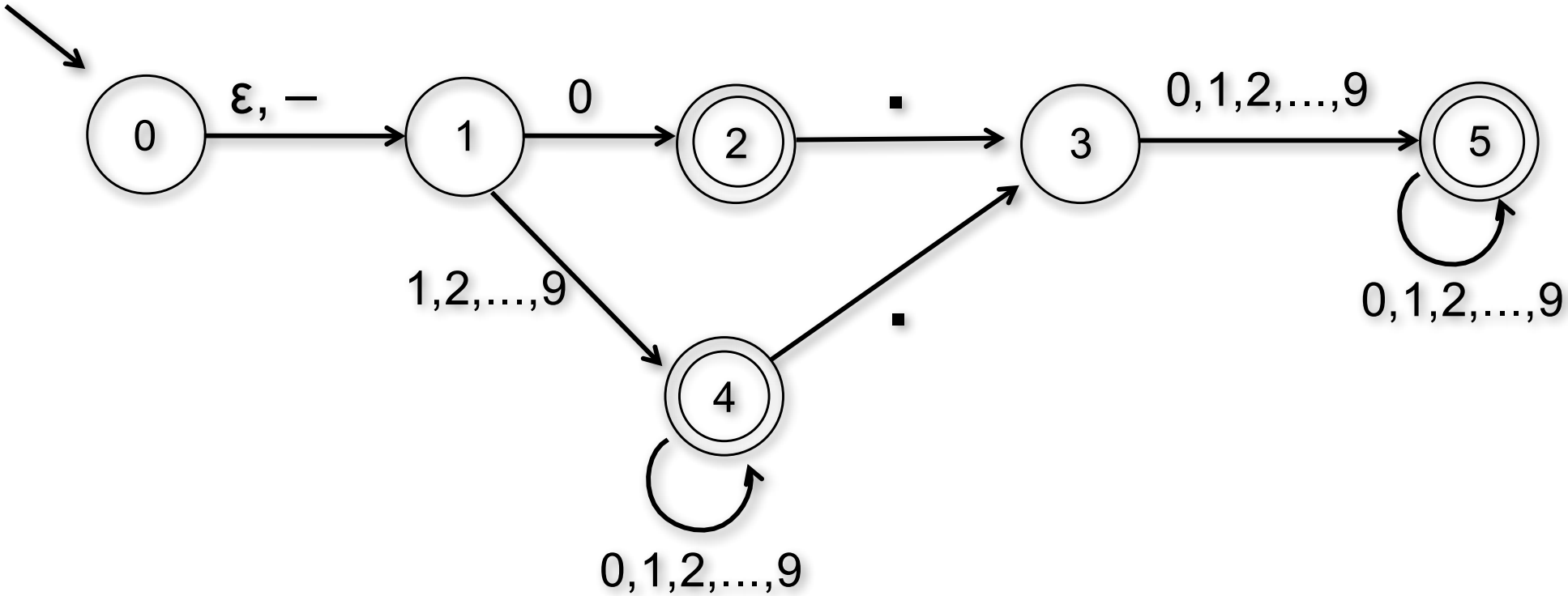
- Every DFA is a NFA so NFAs are at least as powerful as DFAs.
- NFAs prove ability to "guess and verify" which simplifies design and reduces number of states
- Easy proofs of some closure properties

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$\}$

Kleene's Theorem (1951)

Theorem (Informal)

Languages accepted by DFAs are exactly the regular languages.

Proof Strategy:

Kleene's Theorem (1951)

Theorem (Informal)

Languages accepted by DFAs are exactly the regular languages.

Proof Strategy:

1. Regular language $\rightarrow$ NFA
2. NFA $\rightarrow$ DFA
3. DFA $\rightarrow$ regular language

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 IV

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

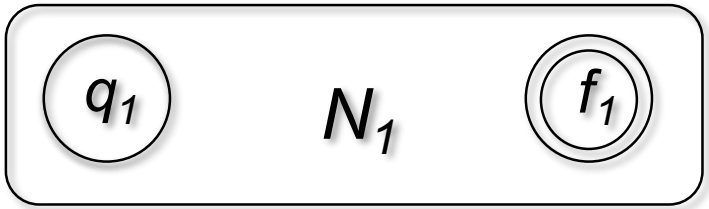
Theorem

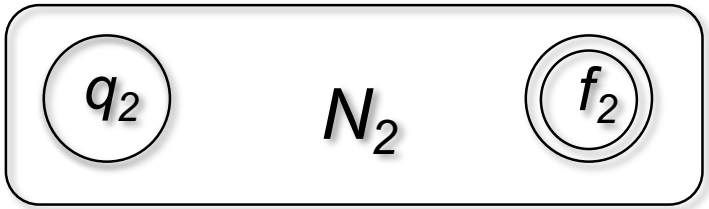
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \cup L(N_2)$.

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

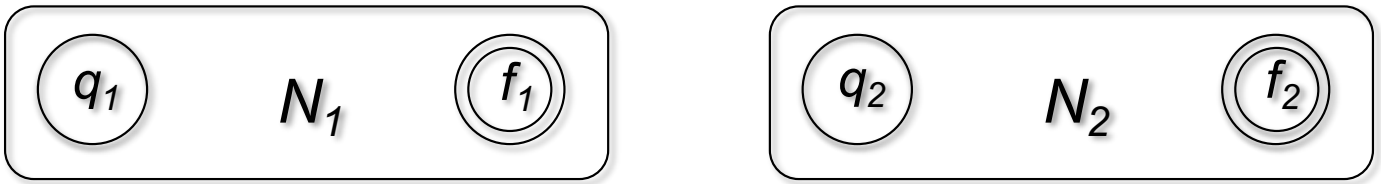
Theorem

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

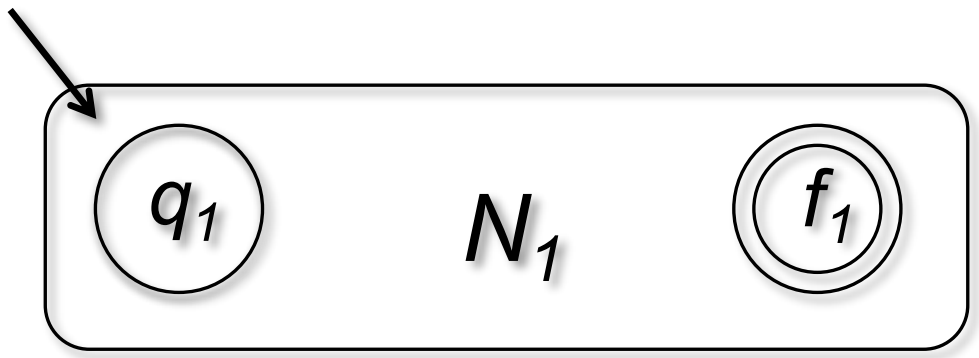
For any two NFAs N_1 and N_2 there is a NFA N such that $L(N) = L(N_1) \bullet L(N_2)$.



Closure under Kleene star

Theorem

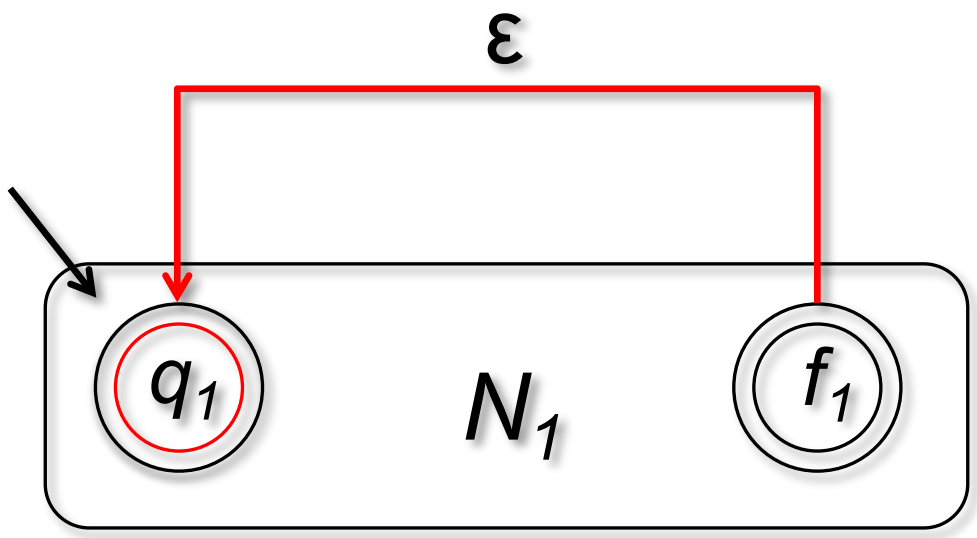
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Closure under Kleene star

Theorem

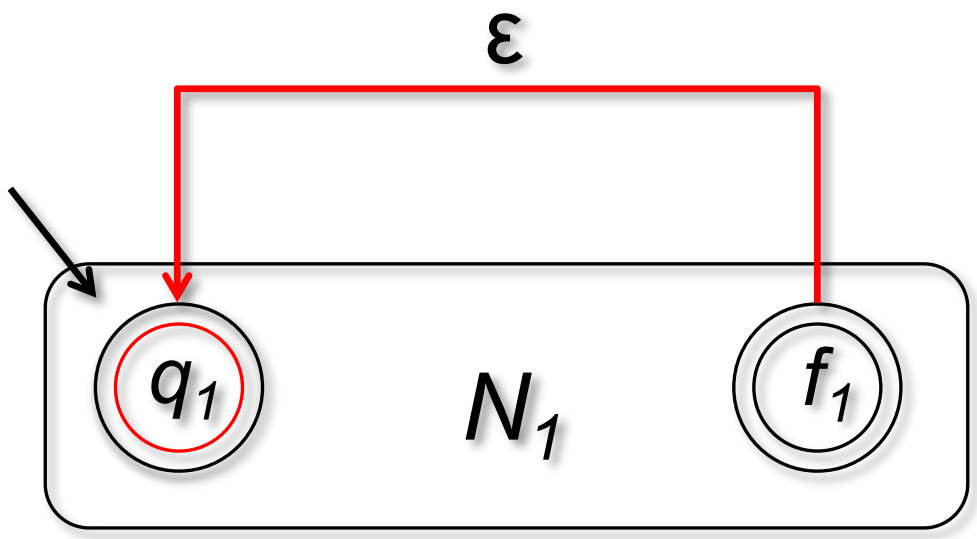
For any NFA N_1 there is a NFA N such that $L(N) = (L(N_1))^*$.



Closure under Kleene star

Theorem

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

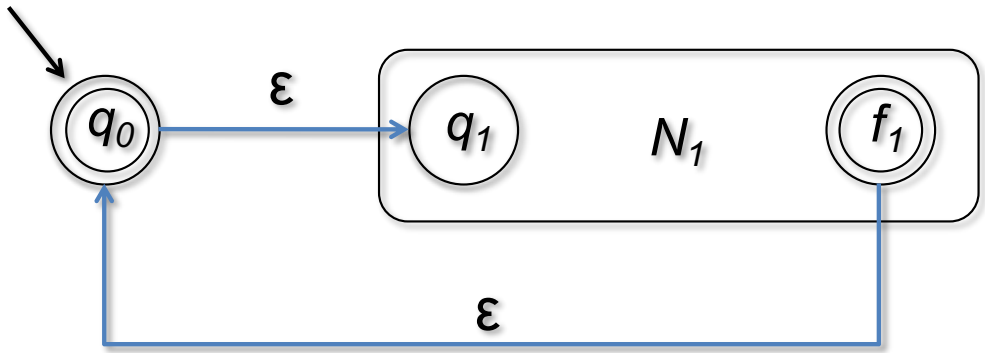


Does not work! Why?

Closure under Kleene star

Theorem

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



Part V

NFAs capture Regular Languages

Regular Languages Recap

Regular Languages	**Regular Expressions**
$\emptyset$ regular	$\emptyset$ denotes $\emptyset$
$\{\epsilon\}$ regular	ϵ denotes $\{\epsilon\}$
$\{a\}$ regular for $a \in \Sigma$	$\mathbf{a}$ denote $\{a\}$
$R_1 \cup R_2$ regular if both are	$\mathbf{r_1} + \mathbf{r_2}$ denotes $R_1 \cup R_2$
$R_1 R_2$ regular if both are	$\mathbf{r_1 r_2}$ denotes $R_1 R_2$
R^* is regular if R is	$\mathbf{r}^*$ 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

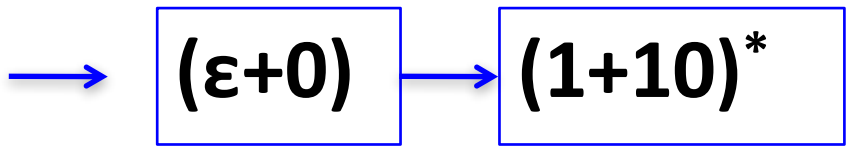
- For every regular expression r show that there is a NFA N such that $L(r) = L(N)$
- Induction on length of r

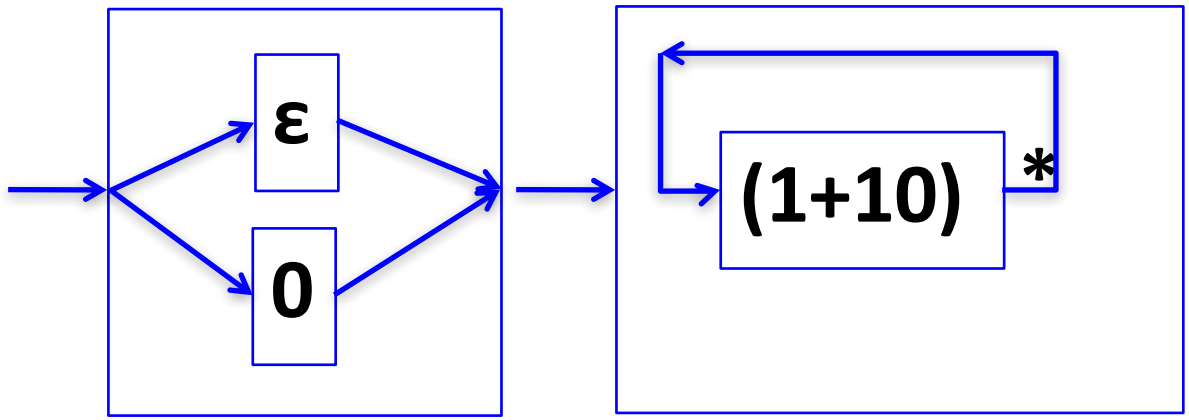
Inductive cases:

- 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

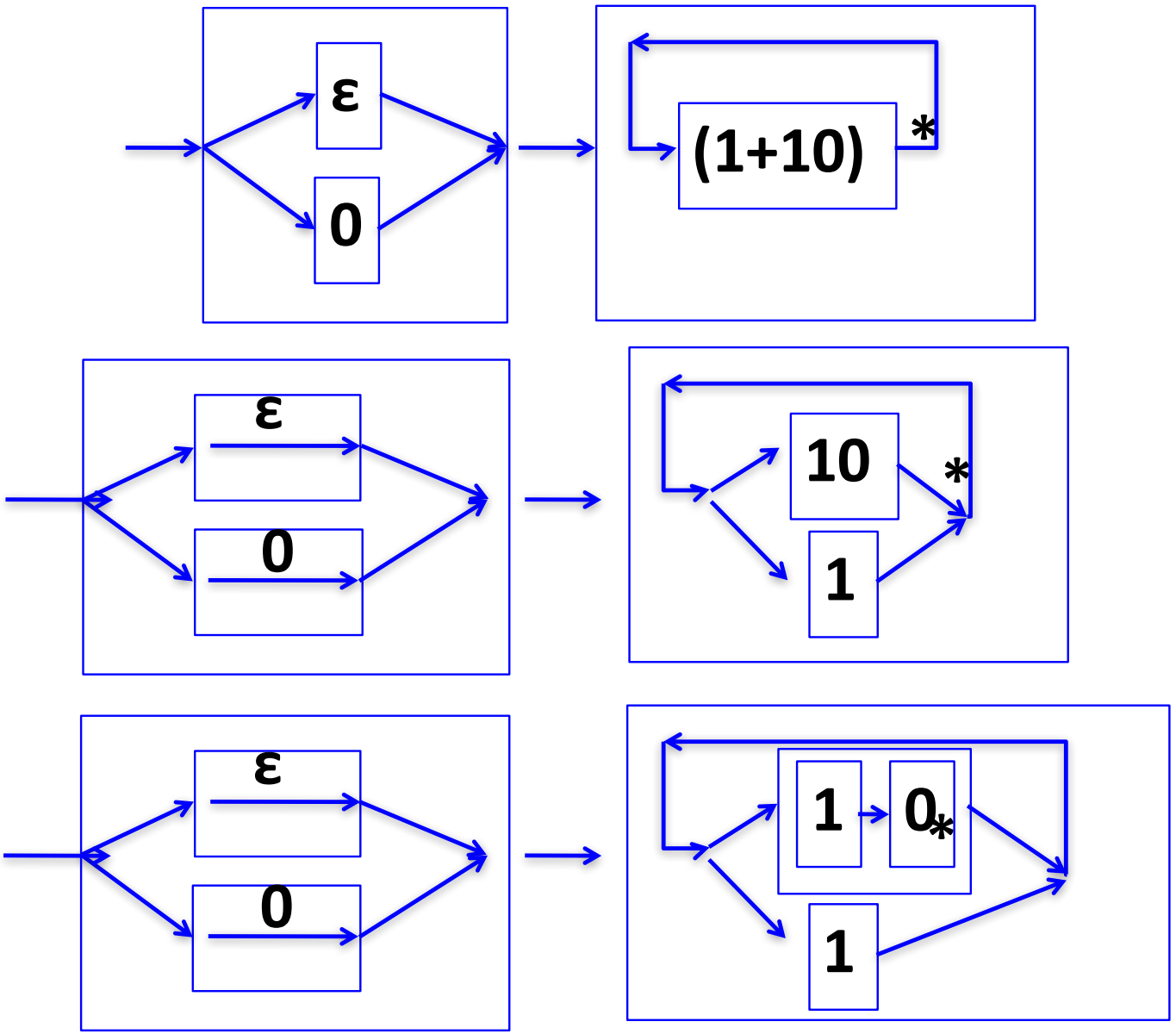
Example

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

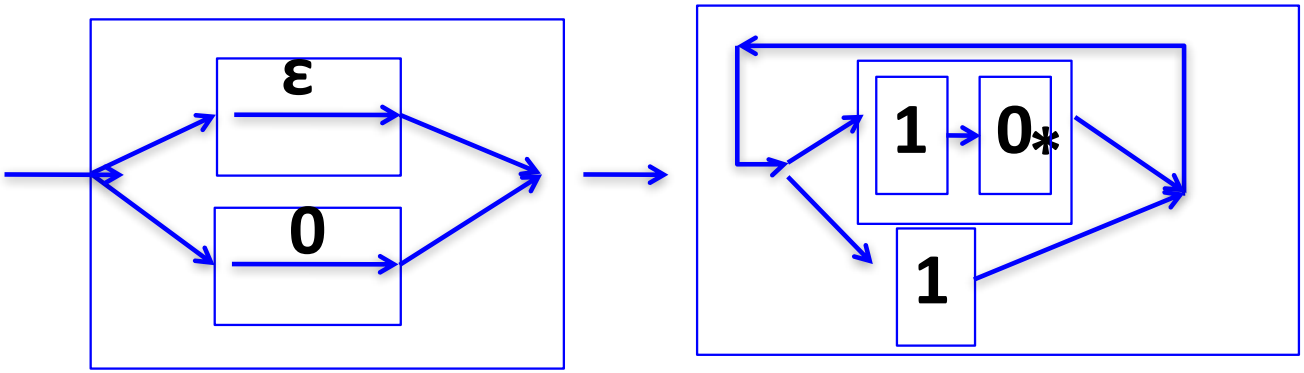


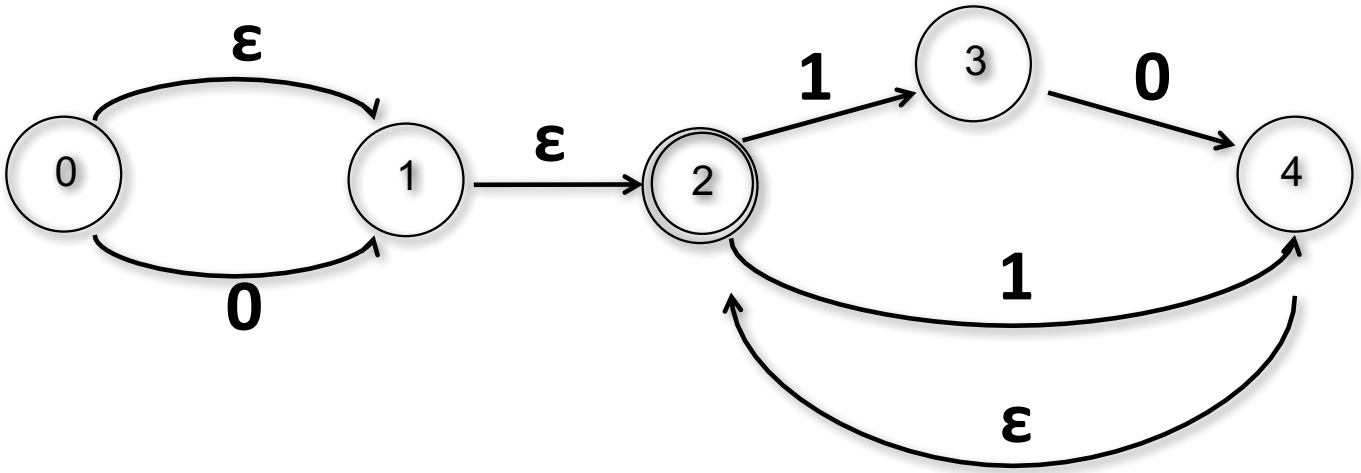


Example



Example





Final NFA simplified slightly to reduce states