NLP 201: Finite State Automata

Jeffrey Flanigan

University of California Santa Cruz

jmflanig@ucsc.edu

Fall 2023

Learning Objectives of NLP 201-203

- Ability to take a description of a model or system and implement it
- Ability to read current papers, understand them, and implement them
- Knowledge of lots of methods for solving problems (including older or alternative methods)
- Intuition into why the methods work and why they were developed
- Theoretical knowledge of why the methods work
- Debugging skills (difficult skill, *very important*)
- Ability to develop new solutions to problems

The plan for the next few lectures

For the next few lectures, we'll start with the simplest models of language.

- Finite state automata (FSAs, **today**)
- Regular languages
- Finite state transducers (FSTs)
- Later: Weighted FSAs and FSTs (WFSAs and WFSTs)

Why learn FSAs?

- Many models (HMMs, n-gram LMs, etc) are special cases of WFSAs
- FSAs and FSTs are used to model morphology
- First step in the formal language hierarchy
- Useful for understanding the modeling capacity of deep NN (for example, A Formal Hierarchy of RNN Architectures `https://arxiv.org/pdf/2004.08500.pdf`)
- Another tool for your toolbox
  - FSAs are incredibly fast

Goals of Today's Lecture

- Understand concepts of **formal languages**
- Understand **operations** on formal languages
- Refresher on proof techniques of **set containment** and **induction**
- Understand two FSAs: **deterministic finite automata** (DFAs), **nondeterministic finite automata** (NFAs) and their equivalence using the **powerset construction**.

Natural languages

- **Natural languages** are languages learned without explicit instruction by children from speakers in their environments
    - e.g., English, Mandarin, Hindi, Cantonese, German
- Natural languages link sounds or visual information (**linguistic forms**) with conceptual/intentional representations (**linguistic functions**)

Linguistic Concept: Grammar

Native speakers have a **grammar**, a (infinite) set of sentences that they accept as being "well-formed" **a sentence can either be in the speaker's grammar, or not in the speaker's grammar**

Example:

- They says "hello."
- They say "hello."

Linguistic Concept: Grammar

Native speakers have a **grammar**, a (infinite) set of sentences that they accept as being "well-formed" **a sentence can either be in the speaker's grammar, or not in the speaker's grammar**

Example:

- * They says "hello." **Linguists use * to indicate ungrammatical.**
- They say "hello."

Linguistic Concept: Grammar

Native speakers have a **grammar**, a (infinite) set of sentences that they accept as being "well-formed" **a sentence can either be in the speaker's grammar, or not in the speaker's grammar**

Example:

- * They says "hello." **Linguists use * to indicate ungrammatical.**
- They say "hello."

What about:

- Colorless green ideas sleep furiously. **Another example: "Colorless green idea sleep furiously." doesn't work, but "A colorless green idea sleeps furiously" does.**

Formal languages

- **Formal languages** are (usually) infinite sets of strings described mathematically
- Formal languages may be used to describe things that are not natural languages, e.g. computer languages

Formal languages

- In this lecture, I will use the terms
  - **alphabet**, the symbols in the language
  - **word**, a string of symbols

Formal languages

- In this lecture, I will use the terms
  - **alphabet**, the symbols in the language
  - **word**, a string of symbols
- Could also have used the terms
  - **vocabulary**, the symbols in the language
  - **sentence**, a string of symbols

Fundamental definitions

Σ is a finite, nonempty **alphabet**

Each symbol is σ in the set is a "letter"

Usually denoted by lowercase letters $a, b, c, \ldots$

A **word** (sentence, string) $\boldsymbol{w} = w_1 w_2 \ldots w_n$

Usually denoted by bold, italic lowercase $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \ldots$

$|\boldsymbol{x}|$ is the **length** of word $\boldsymbol{x}$

ε is the **empty word**, and $|\varepsilon| = 0$

A **language** L is a *set* (finite or infinite) of words from a given alphabet Σ

Fundamental definitions

Σ is a finite, nonempty **alphabet**

Each symbol is σ in the set is a “letter”

Usually denoted by lowercase letters $a, b, c, \ldots$

A **wo**

Us $\boldsymbol{y}, \boldsymbol{z}, \ldots$

$|\boldsymbol{x}|$ is

Careful!

$L_1 = \{\varepsilon\}$ is **not the same** as $L_2 = \emptyset$

ε is the **empty word**, and $|\varepsilon| = 0$

A **language** L is a *set* (finite or infinite) of words from a given alphabet Σ

Formal language operations

Set theoretic **operations** apply to languages

$$L_1 \cup L_2 = \{\boldsymbol{w} \in \Sigma^* \mid \boldsymbol{w} \in L_1 \vee \boldsymbol{w} \in L_2\}$$ **union**

$$L_1 \cap L_2 = \{\boldsymbol{w} \in \Sigma^* \mid \boldsymbol{w} \in L_1 \wedge \boldsymbol{w} \in L_2\}$$ **intersection**

$$\overline{L}\ (= L^C) = \{\boldsymbol{w} \in \Sigma^* \mid \boldsymbol{w} \notin L\}$$ **complementation**

$$\left.\begin{aligned} \overline{(L_1 \cup L_2)} &= \overline{L_1} \cap \overline{L_2} \\ \overline{(L_1 \cap L_2)} &= \overline{L_1} \cup \overline{L_2} \end{aligned}\right\}$$ **de Morgan's laws**

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$

Language concatenation

$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{xy} = aabb$$

Language concatenation

$$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$$

Examples

$$L_1 = \{a, b\} \quad L_2 = \{c, d\} \qquad L_1L_2 =$$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$$

Language concatenation

$$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$$

Examples

$$L_1 = \{a, b\} \quad L_2 = \{c, d\} \qquad L_1L_2 = \{ac, ad, bc, bd\}$$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$

Language concatenation

$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$

Examples

$L_1 = \{a, b\} \quad L_2 = \{c, d\} \qquad L_1L_2 = \{ac, ad, bc, bd\}$

$L.\emptyset =$

$L.\{\varepsilon\} =$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$

Language concatenation

$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$

Examples

$L_1 = \{a, b\} \quad L_2 = \{c, d\} \qquad L_1L_2 = \{ac, ad, bc, bd\}$

$L.\emptyset = \emptyset$

$L.\{\varepsilon\} =$

Concatenation

Concatenation: denoted by a decimal point (but often omitted when meaning is clear from context)

String concatenation

$$\boldsymbol{x} = aa \quad \boldsymbol{y} = bb \quad \boldsymbol{x}.\boldsymbol{y} = \boldsymbol{x}\boldsymbol{y} = aabb$$

Language concatenation

$$L_1.L_2 = L_1L_2 = \{\boldsymbol{x}.\boldsymbol{y} \mid \boldsymbol{x} \in L_1 \wedge \boldsymbol{y} \in L_2\}$$

Examples

$$L_1 = \{a, b\} \quad L_2 = \{c, d\} \qquad L_1L_2 = \{ac, ad, bc, bd\}$$

$$L.\emptyset = \emptyset$$

$$L.\{\varepsilon\} = L$$

Power operation

Power on letters/strings

$$\boldsymbol{w}^n = \boldsymbol{w}.\boldsymbol{w}^{n-1} \quad \boldsymbol{w}^0 = \varepsilon$$

$$\boldsymbol{w} = ab$$

$$\boldsymbol{w}^2 = abab$$

$$\dots$$

Power on languages

$$L^n = L.L^{n-1} \quad L^0 = \{\varepsilon\}$$

Power operation

Power on letters/strings

$$\boldsymbol{w}^n = \boldsymbol{w}.\boldsymbol{w}^{n-1} \quad \boldsymbol{w}^0 = \varepsilon$$

$$\boldsymbol{w} = ab$$

$$\boldsymbol{w}^2 = abab$$

$$\dots$$

Power on languages

$$L^n = L.L^{n-1} \quad L^0 = \{\varepsilon\}$$

The Kleene *

Kleene * on letters

$$a^* = \bigcup_{i=0}^{\infty} a^i = \{\varepsilon, a, aa, aaa, \ldots\}$$

The Kleene *

Kleene * on letters

$$a^* = \bigcup_{i=0}^{\infty} a^i = \{\varepsilon, a, aa, aaa, \ldots\}$$

Kleene * on languages

$$\begin{aligned} L^* &= \bigcup_{i=0}^{\infty} L^i = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \ldots \\ &= \{\boldsymbol{w} \mid \exists i \text{ s.t. } \boldsymbol{w} \in L^i\} \end{aligned}$$

The Kleene *

Kleene * on letters

$$a^* = \bigcup_{i=0}^{\infty} a^i = \{\varepsilon, a, aa, aaa, \ldots\}$$

Kleene * on languages

$$L^* = \bigcup_{i=0}^{\infty} L^i = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \ldots$$

$$= \{\boldsymbol{w} \mid \exists i \text{ s.t. } \boldsymbol{w} \in L^i\}$$

Important notation

$\Sigma^* =$ set of all finite words over Σ

$\Sigma^i =$ set of all words of length i over Σ

The Kleene *

Kleene * on letters

$$a^* = \bigcup^{\infty} a^i = \{\varepsilon, a, aa, aaa, \ldots\}$$

Kleen

Notational variant, the **Kleene** +

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L \cup L^2 \cup L^3 \cup \ldots$$

Does not *add* the empty string ε

Important notation

$\Sigma^* =$ set of all finite words over Σ

$\Sigma^i =$ set of all words of length i over Σ

Finite-State Automata

Definition: DFA

A **deterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

Σ is a finite **alphabet**

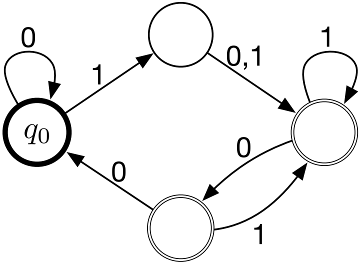
$\delta : Q \times \Sigma \to Q$ is the **transition function**

$q_0 \in Q$ is the **start (initial) state**

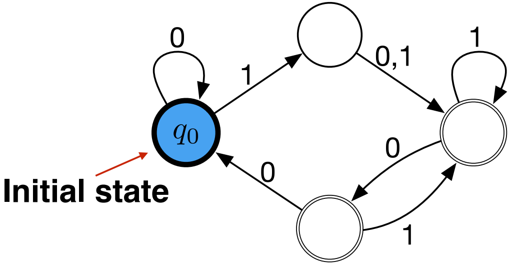
$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

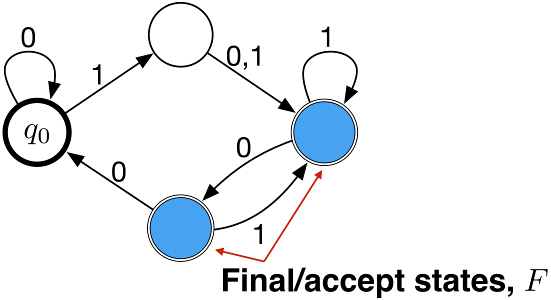
DFAs



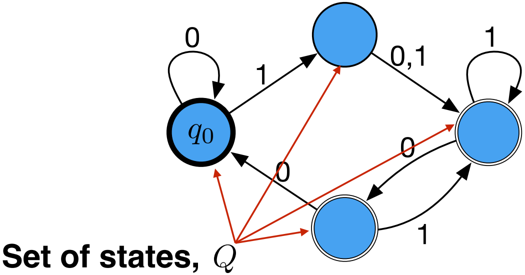
DFAs



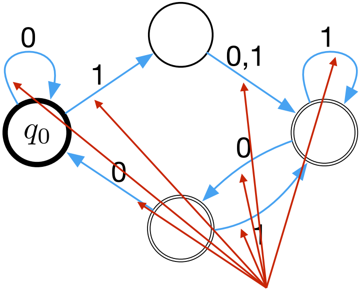
DFAs



DFAs



DFAs



Transition function (“edges”), δ

Acceptance

- M is a DFA with alphabet Σ
- An input sentence (word / string) ***w*** is a sequence $w_1 w_2 w_3 \ldots w_n$ where each $w_i \in \Sigma$
- M accepts ***w*** if after starting in q_0 and reading ***w*** it ends in an accept state, i.e., if there is a sequence of states $s_0, s_1, \ldots, s_n$ such that

$$s_0 = q_0$$
$$s_i = \delta(s_{i-1}, w_i)$$
$$s_n \in F$$

Acceptance and Analysis

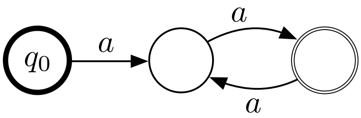
- In many problems in natural language processing, we want to analyze a strings in terms of underlying operations.

- With finite state models, this usually corresponds to the question: what was the sequence of states $s_0,s_1,\ldots,s_n$ taken to produce some $\boldsymbol{w}$?

Example

The following DFA accepts the language

$L = \{a^{2n} \mid n \geq 1\}$



Definition: DFA

A **deterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

Σ is a finite **alphabet**

$\delta : Q \times Q \to \Sigma$ is the **transition function**

$q_0 \in Q$ is the **start (initial) state**

$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

Definition: NFA

A **nondeterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

Σ is a finite **alphabet**

~~$\delta : Q \times Q \to \Sigma$ is a **transition function**~~

$\delta : Q \times \Sigma \to 2^Q$ is the **transition relation**

$q_0 \in Q$ is the **start (initial) state**

$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

Definition: NFA

A **nondeterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

An NFA accepts a word ***w*** if there exists a computation path that ends in a final state.

$\delta : Q \times \Sigma \to 2^Q$ is the **transition relation**

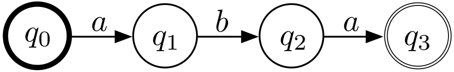
$q_0 \in Q$ is the **start (initial) state**

$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

Remarks

- A DFA is required to have a **complete** transition function, whereas an NFA can have an **incomplete** transition function
- This is useful for specifying simple language generating automata, e.g. “linear chain” automata

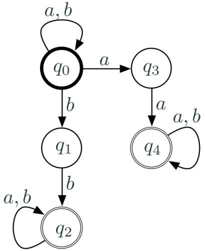


$$L(M) = \{aba\}$$

Board Work: NFA Example

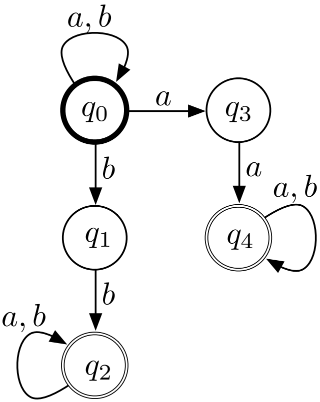
(5 Min) For the following NFA, does it accept the string $\mathbf{w} = abaa$? What sequence of states does it go through?

Can you describe in English the strings that it accepts?

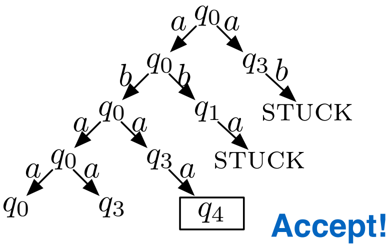


(3 Min) Discuss with a partner.

Example: NFA



Computation on $\boldsymbol{w} = abaa$



Definition: NFA

A **nondeterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

Σ is a finite **alphabet**

~~$\delta : Q \times Q \to \Sigma$ is a **transition function**~~

$\delta : Q \times \Sigma \to 2^Q$ is the **transition relation**

$q_0 \in Q$ is the **start (initial) state**

$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

Definition: NFA

A **nondeterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

An NFA accepts a word ***w*** if there exists a computation path that ends in a final state.

$\delta : Q \times \Sigma \to 2^Q$ is the **transition relation**

$q_0 \in Q$ is the **start (initial) state**

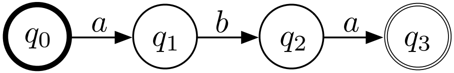
$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

Remarks

- A DFA is required to have a **complete** transition function, whereas an NFA can have an **incomplete** transition function

- This is useful for specifying simple language generating automata, e.g. “linear chain” automata



$$L(M) = \{aba\}$$

DFAs vs NFAs

- For each state, a DFA has exactly one outgoing edge for each symbol in Σ.
- Otherwise, it's a NFA

DFA & NFA Equivalence

Theorem. For every NFA A there exists a DFA A' s.t. $L(A) = L(A')$.

Regular Languages

A language $L \subseteq \Sigma^*$ is **regular** if there exists an FSA M such that $L(M) = L$

Regular Languages

A language $L \subseteq \Sigma^*$ is **regular** if there exists an FSA M such that $L(M) = L$

Example regular languages:

$$L = \Sigma^*$$

$$L = \emptyset$$

$$L = \{\varepsilon\}$$

Regular Languages

A language $L \subseteq \Sigma^*$ is **regular** if there exists an FSA M such that $L(M) = L$

Example regular languages:

$$L = \Sigma^*$$

$$L = \emptyset$$

$$L = \{\varepsilon\}$$

Example non-regular languages (more later):

$$L = \{a^n b^n \mid n \geq 0\}$$
$$L = \{\boldsymbol{w} \mid \boldsymbol{w} \text{ is a grammatical sentence of English}\}$$

Generalization

It is helpful to generalize the transition function of DFAs in terms of **words** (instead of single symbols).

$$\delta(q, a) = q'$$

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

$$\hat{\delta}(q, \varepsilon) = q$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \delta(\hat{\delta}(q, \boldsymbol{x}), \sigma)$$

Generalization

It is helpful to generalize the transition function of DFAs in terms of **words** (instead of single symbols).

$$\delta(q, a) = q'$$

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

$$\hat{\delta}(q, \varepsilon) = q$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \delta(\hat{\delta}(q, \boldsymbol{x}), \sigma)$$

Since $\hat{\delta}(q, \sigma) = \delta(q, \sigma)$, we will not distinguish between these functions.

Generalization

It is helpful to generalize the transition function of DFAs in terms of **words** (instead of single symbols).

$$\delta(q, a) = q'$$

Formal definition of $L(M)$

$$L(M) = \left\{ \boldsymbol{w} \in \Sigma^* \mid \hat{\delta}(q_0, \boldsymbol{w}) \in F \right\}$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \delta(\hat{\delta}(q, \boldsymbol{x}), \sigma)$$

Since $\hat{\delta}(q, \sigma) = \delta(q, \sigma)$, we will not distinguish between these functions.

Generalization

As with DFAs, it is helpful to generalize the transition function of NFAs in terms of **words** (instead of single symbols).

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \bigcup_{q' \in \hat{\delta}(q, \boldsymbol{x})} \delta(q', \sigma)$$

Generalization

As with DFAs, it is helpful to generalize the transition function of NFAs in terms of **words** (instead of single symbols).

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \bigcup_{q' \in \hat{\delta}(q, \boldsymbol{x})} \delta(q', \sigma)$$

We also generalize in terms of **sets of states**:

$$P \in 2^Q$$

$$\delta(P, \boldsymbol{x}) = \bigcup_{p \in P} \delta(p, \boldsymbol{x})$$

Generalization

As with DFAs, it is helpful to generalize the transition function of NFAs in terms of **words** (instead of single symbols).

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \bigcup_{q' \in \hat{\delta}(q, \boldsymbol{x})} \delta(q', \sigma)$$

We also generalize in terms of **sets of states**:

$$P \in 2^Q$$

$$\delta(P, \boldsymbol{x}) = \bigcup_{p \in P} \delta(p, \boldsymbol{x})$$

Again, we will not distinguish between the generalized form of the function and the basic form.

Generalization

As with DFAs, it is helpful to generalize the transition function of NFAs in terms of **words** (instead of single symbols).

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

$$\hat{\delta}(q, \boldsymbol{x}\sigma) = \bigcup \delta(q', \sigma)$$

> Formal definition of $L(M)$
>
> $$L(M) = \{\boldsymbol{w} \in \Sigma^* \mid \delta(q_0, \boldsymbol{w}) \cap F \neq \emptyset\}$$

We

$$\delta(P, \boldsymbol{x}) = \bigcup_{p \in P} \delta(p, \boldsymbol{x})$$

Again, we will not distinguish between the generalized form of the function and the basic form.

DFA & NFA Equivalence

Theorem. For every NFA A there exists a DFA A' s.t. $L(A) = L(A')$.

Proofs

- Proofs in formal language theory often use **set containment** and **induction**

Proofs

- Proofs in formal language theory often use **set containment** and **induction**

- **Set containment**
  To prove that $(L_1 \cup L_2).L_3 = L_1.L_3 \cup L_2.L_3$

  one can show that (i)
  $\mathbf{w} \in (L_1 \cup L_2).L_3 \implies \mathbf{w} \in L_1.L_3 \cup L_2.L_3$

  and the other direction that (ii)
  $\mathbf{w} \in L_1.L_3 \cup L_2.L_3 \implies \mathbf{w} \in (L_1 \cup L_2).L_3$

Proofs

- Proofs in formal language theory often use **set containment** and **induction**

- **Induction**
  Induction is usually on the length of the word, i.e. to prove P is true for all $\mathbf{w} \in \mathbf{L}$, we:
  1. show P is true for words length 0 or 1
  2. hypothesize that P is true for all words $\leq$ length $n - 1$
  3. prove that if P is true for all words $\leq$ length $n - 1$, then P is true for for all words of length n

Board work: Proof by induction

On your whiteboards, take 5 min to do 1. and 3.
Then discuss with a partner for 3 min

- **Induction example**
  Prove that $\forall n \geq 0$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

1. base: verify for $n = 0$
2. hypothesis: assume true for $n \leq k - 1$
3. inductive step: prove for $n = k$

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Proofs

- **Induction example**
  Prove that $\forall n \geq 0$
  $$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

  1. base: $n = 0$: $2^0 = 1 = 2^{0+1} - 1$
  2. hypothesis: assume true for $n \leq k - 1$
  3. inductive step: prove for $n = k$

$$\begin{aligned}\sum_{i=0}^{k} 2^i &= 2^k + \sum_{i=0}^{k-1} 2^i \\ &= 2^k + 2^k - 1 \quad \text{(by inductive hypothesis)} \\ &= 2(2^k) - 1 \\ &= 2^{k+1} - 1 \quad \square\end{aligned}$$

DFA & NFA Equivalence

Theorem. For every NFA A there exists a DFA A' s.t. $L(A) = L(A')$.

DFA & NFA Equivalence

Theorem. For every NFA A there exists a DFA A' s.t. $L(A) = L(A')$.

Proof. *This is a constructive proof.* That is, given an NFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ we construct a DFA $A' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$ s.t. $L(A) = L(A')$

DFA & NFA Equivalence

Theorem. For every NFA A there exists a DFA A' s.t. $L(A) = L(A')$.

Proof. *This is a constructive proof.* That is, given an NFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ we construct a DFA $A' = \langle Q', \Sigma', \delta', q_0', F' \rangle$ s.t. $L(A) = L(A')$

Powerset construction.

$\Sigma' = \Sigma$

$Q' = 2^Q$ Constructed DFA states are **sets** of NFA states

$q_0' = \{q_0\}$

$F' = \{A \in Q' \mid A \cap F \neq \emptyset\}$

DFA & NFA Equivalence

Powerset construction.

$\Sigma' = \Sigma$

$Q' = 2^Q$ Constructed DFA states are **sets** of NFA states

$q_0' = \{q_0\}$

$F' = \{A \in Q' \mid A \cap F \neq \emptyset\}$

DFA & NFA Equivalence

Powerset construction.

$\Sigma' = \Sigma$

$Q' = 2^Q$ Constructed DFA states are **sets** of NFA states

$q_0' = \{q_0\}$

$F' = \{A \in Q' \mid A \cap F \neq \emptyset\}$

Transition function.

DFA & NFA Equivalence

Powerset construction.

$\Sigma' = \Sigma$

$Q' = 2^Q$ Constructed DFA states are **sets** of NFA states

$q_0' = \{q_0\}$

$F' = \{A \in Q' \mid A \cap F \neq \emptyset\}$

Transition function.

$\delta'(\emptyset, \sigma) = \emptyset \ \forall \sigma \in \Sigma'$ "Failure state"

DFA & NFA Equivalence

Powerset construction.

$\Sigma' = \Sigma$

$Q' = 2^Q$ Constructed DFA states are **sets** of NFA states

$q'_0 = \{q_0\}$

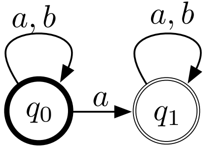
$F' = \{A \in Q' \mid A \cap F \neq \emptyset\}$

Transition function.

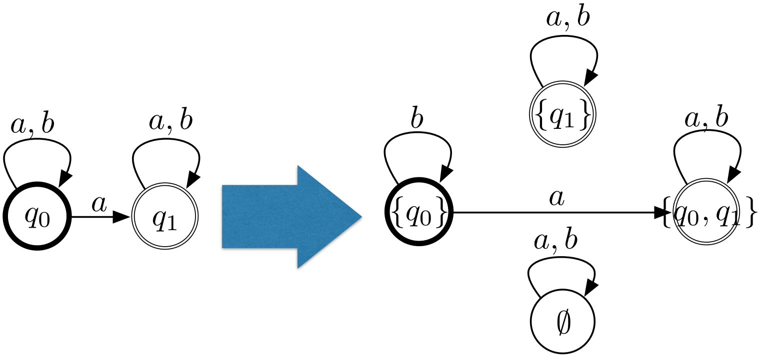
$\delta'(\emptyset, \sigma) = \emptyset \ \forall \sigma \in \Sigma'$ "Failure state"

$$\delta'(\{q_1, q_2, \ldots, q_i\}, \sigma) = \bigcup_{q \in \{q_1, q_2, \ldots, q_i\}} \delta(q, \sigma)$$

Example



Example



DFA & NFA Equivalence

It remains to show:

A' is DFA $\qquad$ $L(A) = L(A')$

DFA & NFA Equivalence

It remains to show:

A' is DFA $\qquad\qquad L(A) = L(A')$

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

DFA & NFA Equivalence

It remains to show:

A' is DFA $\qquad\qquad L(A) = L(A')$

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$
$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

Proof of lemma. By induction on $|\boldsymbol{w}|$

DFA & NFA Equivalence

It remains to show:

A' is DFA $\qquad\qquad L(A) = L(A')$

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$
$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

Proof of lemma. By induction on $|\boldsymbol{w}|$

(i) base: $|\boldsymbol{w}| = 0 \quad \boldsymbol{w} = \varepsilon :\ \delta'(q_0', \varepsilon) = \{q_0\}$

$$\delta(q_0, \varepsilon) = \{q_0\}$$

DFA & NFA Equivalence

It remains to show:

A' is DFA $\qquad L(A) = L(A')$

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$
$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

Proof of lemma. By induction on $|\boldsymbol{w}|$

(i) base: $|\boldsymbol{w}| = 0 \quad \boldsymbol{w} = \varepsilon :\ \delta'(q_0', \varepsilon) = \{q_0\}$

$$\delta(q_0, \varepsilon) = \{q_0\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

(iii) inductive step: let $\boldsymbol{w} = \boldsymbol{x}\sigma$ s.t. $|\boldsymbol{w}| = n + 1$

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

(iii) inductive step: let $\boldsymbol{w} = \boldsymbol{x}\sigma$ s.t. $|\boldsymbol{w}| = n + 1$

$$\delta'(q_0', \boldsymbol{w}) = \delta'(\{q_0\}, \boldsymbol{x}\sigma) = \delta'(\delta'(\{q_0\}, \boldsymbol{x}), \sigma)$$

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

(iii) inductive step: let $\boldsymbol{w} = \boldsymbol{x}\sigma$ s.t. $|\boldsymbol{w}| = n+1$

$$\delta'(q_0', \boldsymbol{w}) = \delta'(\{q_0\}, \boldsymbol{x}\sigma) = \delta'(\delta'(\{q_0\}, \boldsymbol{x}), \sigma)$$

use inductive hypothesis:

$$\delta'(\{q_0\}, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\} \iff \delta(q_0, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\}$$

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$

$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

(iii) inductive step: let $\boldsymbol{w} = \boldsymbol{x}\sigma$ s.t. $|\boldsymbol{w}| = n + 1$

$$\delta'(q_0', \boldsymbol{w}) = \delta'(\{q_0\}, \boldsymbol{x}\sigma) = \delta'(\delta'(\{q_0\}, \boldsymbol{x}), \sigma)$$

use inductive hypothesis:

$$\delta'(\{q_0\}, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\} \iff \delta(q_0, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\}$$

$$\delta'(\{p_1, p_2, \ldots, p_k\}, \sigma) = \{r_1, r_2, \ldots, r_j\} \iff \delta(\{p_1, p_2, \ldots, p_k\}, \sigma) = \{r_1, r_2, \ldots, r_j\}$$

DFA & NFA Equivalence

Lemma. For every $\boldsymbol{w} \in \Sigma^*$,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\} \iff$$
$$\delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_k\}$$

(ii) hypothesis: assume true for all $\boldsymbol{w}$ s.t. $|\boldsymbol{w}| \leq n$.

(iii) inductive step: let $\boldsymbol{w} = \boldsymbol{x}\sigma$ s.t. $|\boldsymbol{w}| = n + 1$

$$\delta'(q_0', \boldsymbol{w}) = \delta'(\{q_0\}, \boldsymbol{x}\sigma) = \delta'(\delta'(\{q_0\}, \boldsymbol{x}), \sigma)$$

use inductive hypothesis:

$$\delta'(\{q_0\}, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\} \iff \delta(q_0, \boldsymbol{x}) = \{p_1, p_2, \ldots, p_k\}$$

$$\delta'(\{p_1, p_2, \ldots, p_k\}, \sigma) = \{r_1, r_2, \ldots, r_j\} \iff \delta(\{p_1, p_2, \ldots, p_k\}, \sigma) = \{r_1, r_2, \ldots, r_j\}$$

$$\delta'(q_0', \boldsymbol{w}) = \{r_1, r_2, \ldots, r_j\} = \delta(q_0, \boldsymbol{w}) \blacksquare$$

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad$ $L(A) = L(A')$

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad\qquad L(A) = L(A')$

Proof of theorem.

By definition, A' is a complete function, so it is a DFA

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad$ $L(A) = L(A')$

Proof of theorem.

By definition, A' is a complete function, so it is a DFA

From the lemma,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\} \iff \delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\}$$

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad$ $L(A) = L(A')$

Proof of theorem.

By definition, A' is a complete function, so it is a DFA

From the lemma,

$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\} \iff \delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\}$

$\{p_1, p_2, \ldots, p_j\} \in F' \iff \{p_1, p_2, \ldots, p_j\} \cap F \neq \emptyset$

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad\qquad L(A) = L(A')$

Proof of theorem.

By definition, A' is a complete function, so it is a DFA

From the lemma,

$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\} \iff \delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\}$

$\{p_1, p_2, \ldots, p_j\} \in F' \iff \{p_1, p_2, \ldots, p_j\} \cap F \neq \emptyset$

thus, $\boldsymbol{w} \in L(A') \iff \boldsymbol{w} \in L(A)$

DFA & NFA Equivalence

It still remains to show:

A' is DFA $\qquad L(A) = L(A')$

Proof of theorem.

By definition, A' is a complete function, so it is a DFA

From the lemma,

$$\delta'(q_0', \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\} \iff \delta(q_0, \boldsymbol{w}) = \{p_1, p_2, \ldots, p_j\}$$

$$\{p_1, p_2, \ldots, p_j\} \in F' \iff \{p_1, p_2, \ldots, p_j\} \cap F \neq \emptyset$$

thus, $\boldsymbol{w} \in L(A') \iff \boldsymbol{w} \in L(A)$

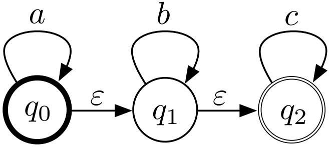
so, $L(A') = L(A)$ ■

Remarks

- NFAs therefore only accept regular languages
- NFAs and DFAs can be used interchangeably

NFA with Epsilons

A **NFA with ε-transitions** is an NFA that may change states without reading an input symbol.



NFA with Epsilons

A **nondeterministic finite automaton** is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

Q is a finite **set of states**

Σ is a finite **alphabet**

~~$\delta : Q \times \Sigma \to 2^Q$ is the **transition relation**~~

$\delta : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ is the **transition relation**

$q_0 \in Q$ is the **start (initial) state**

$F \subseteq Q$ is the **set of final (accept) states**

$L(M) \subseteq \Sigma^*$ is **the language of** M, i.e. the set of strings M accepts

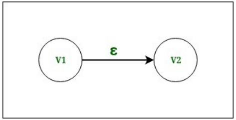
NFA& ε-NFA Equivalence

Theorem. For every NFA A with epsilon moves there is an equivalent NFA A' without, s.t. $L(A) = L(A')$

Conversion of ϵ-NFA to NFA

First do ϵ closure (add all implicit ϵ moves)

Then, for every epsilon move:



1. For all moves to any state that start from V2, duplicate the moves but with the start state V1.
2. If V1 is a start state, make V2 a start state as well
3. If V2 is a final state, make V1 a final state as well
4. Remove the ϵ-move

Regular Expression

A regular expression is a way of describing the languages accepted by FSAs.

Defined recursively:

1. $\emptyset$ is an RE denoting the empty set
2. ε is an RE denoting the set $\{\varepsilon\}$
3. for each $a \in \Sigma$, a is a RE denoting $\{a\}$
4. If `r` and `s` are REs denoting the languages R and S

   `(r|s)` denotes $R \cup S$

   `(rs)` denotes $R.S$

   `r*` denotes R^*

Precedence means parentheses can sometimes be omitted: `*.|`

Examples

`(0|1)*` denotes all finite words over $\Sigma = \{0, 1\}$

`0*|1*` denotes all finite words containing only 0's and 1's

Board Work: Regex Examples

(5 Min) For the following languages, write it's regular expression

- Strings over $\{a, b\}^*$ that start with an a
- Strings over $\{a, b\}^*$ that contain ab or ba

(3 Min) Discuss with a partner.

Regex in practice

Most programming languages, grep, etc use a slightly different syntax. By default, the regex only needs to match a substring.

Character	Meaning	Example
*	Match **zero, one or more** of the previous	`Ah*` matches "`Ahhhhh`" or "`A`"
?	Match **zero or one** of the previous	`Ah?` matches "`Al`" or "`Ah`"
+	Match **one or more** of the previous	`Ah+` matches "`Ah`" or "`Ahhh`" but not "`A`"
\	Used to **escape** a special character	`Hungry\?` matches "`Hungry?`"
.	Wildcard character, matches **any** character	`do.*` matches "`dog`", "`door`", "`dot`", etc.
( )	**Group** characters	See example for |
[ ]	Matches a **range** of characters	`[cbf]ar` matches "car", "bar", or "far" `[0-9]+` matches any positive integer `[a-zA-Z]` matches ascii letters a-z (uppercase and lower case) `[^0-9]` matches any character not 0-9.
|	Matche previous **OR** next character/group	`(Mon|Tues)day` matches "Monday" or "Tuesday"
{ }	Matches a specified **number of occurrences** of the previous	`[0-9]{3}` matches "315" but not "31" `[0-9]{2,4}` matches "12", "123", and "1234" `[0-9]{2,}` matches "1234567..."
^	**Beginning** of a string. Or within a character range `[]` negation.	`^http` matches strings that begin with http, such as a url. `[^0-9]` matches any character not 0-9.
$	**End** of a string.	`ing$` matches "exciting" but not "ingenious"

REs and ε-NFAs

Theorem. For every RE $\mathtt{r}$ there is an ε-NFA A s.t. $L(\mathtt{r}) = L(A)$

REs and ε-NFAs

Theorem. For every RE $\texttt{r}$ there is an ε-NFA s.t. $L(\texttt{r}) = L(A)$

Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

REs and ε-NFAs

Theorem. For every RE $\texttt{r}$ there is an ε-NFA s.t. $L(\texttt{r}) = L(A)$

Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

Base cases. **r** has 0 operators

REs and ε-NFAs

Theorem. For every RE $\mathtt{r}$ there is an ε-NFA s.t. $L(\mathtt{r}) = L(A)$

Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

Base cases. **r** has 0 operators

$\mathtt{r} = \emptyset$

REs and ε-NFAs

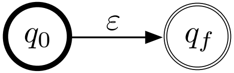
Theorem. For every RE $\mathtt{r}$ there is an ε-NFA s.t. $L(\mathtt{r}) = L(A)$

Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

Base cases. **r** has 0 operators

$\mathtt{r} = \emptyset$

$\mathtt{r} = \varepsilon$



REs and ε-NFAs

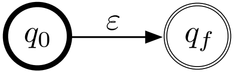
Theorem. For every RE $\mathtt{r}$ there is an ε-NFA s.t.
$L(\mathtt{r}) = L(A)$

Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

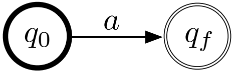
Base cases. **r** has 0 operators

$\mathtt{r} = \emptyset$

$\mathtt{r} = \varepsilon$



$\mathtt{r} = \mathtt{a}$



REs and ε-NFAs

Theorem. For every RE $\mathtt{r}$ there is an ε-NFA s.t.
$L(\mathtt{r}) = L(A)$

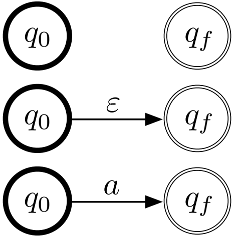
Proof. We will construct *A* compositionally using induction on the number of operators in **r**.

Base cases. **r** has 0 operators

$\mathtt{r} = \emptyset$

$\mathtt{r} = \varepsilon$

$\mathtt{r} = \mathtt{a}$



Note: we assume there is **exactly one** final state.

REs and ε-NFAs

Inductive step. We assume hypothesis is true for all REs with $\leq n$ operations, and then prove is true for $n+1$ operations.

REs and ε-NFAs

Inductive step. We assume hypothesis is true for all REs with ≤*n* operations, and then prove is true for *n*+1 operations.

There are three cases to be dealt with:

(1) $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

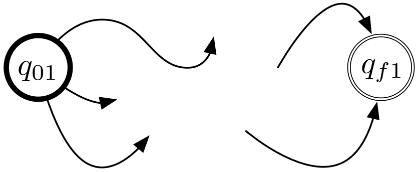
(2) $\mathtt{r} = \mathtt{r}_1\mathtt{r}_2$

(3) $\mathtt{r} = \mathtt{r}_1\texttt{*}$

Case 1: $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

By the inductive hypothesis, there are two epsilon NFAs A_1 and A_2.

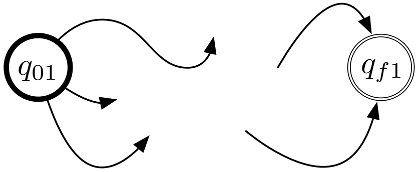
A_1



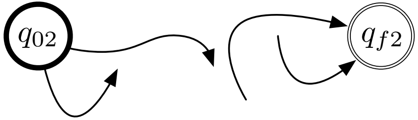
Case 1: $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

By the inductive hypothesis, there are two epsilon NFAs A_1 and A_2.

A_1

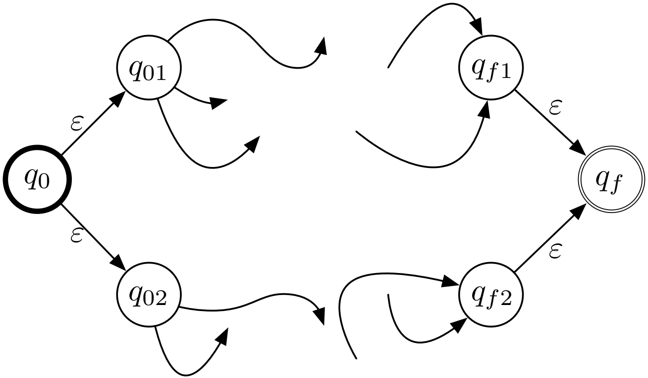


A_2



Case 1: $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

By the inductive hypothesis, there are two epsilon NFAs A_1 and A_2. **Construct the following** A.



Case 1: $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

Formally, if $A_1 = \langle Q_1, \Sigma, \delta_1, q_{01}, \{q_{f1}\} \rangle$

$$A_2 = \langle Q_2, \Sigma, \delta_2, q_{02}, \{q_{f2}\} \rangle$$

then,

$A = \langle Q_1 \cup Q_2 \cup \{q_0\} \cup \{q_f\}, \Sigma, \delta, q_0, \{q_f\} \rangle$

$$\begin{aligned}
\delta(q_0, \varepsilon) &= \{q_{01}, q_{02}\} \\
\delta(q_{f1}, \varepsilon) &= \{q_f\} \\
\delta(q_{f2}, \varepsilon) &= \{q_f\} \\
\delta(q, \sigma) &= \delta_1 q, \sigma \quad \forall q \in Q_1 - \{q_{f1}\}, \sigma \in \Sigma \cup \{\varepsilon\} \\
\delta(q, \sigma) &= \delta_2 q, \sigma \quad \forall q \in Q_2 - \{q_{f2}\}, \sigma \in \Sigma \cup \{\varepsilon\}
\end{aligned}$$

Case 1: $\mathtt{r} = \mathtt{r}_1 \mid \mathtt{r}_2$

It remains to show that $L(A) = L(A_1) \cup L(A_2)$

How to do this? Set containment.

Cases 2 & 3

- Strategy for showing this proceeds as with Case 1
- Refer to textbook for details.