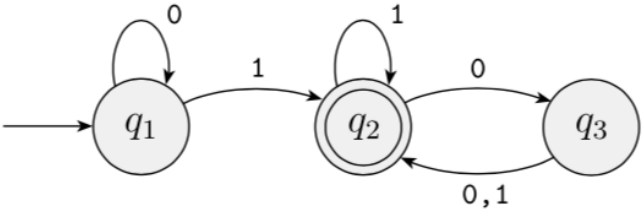
Administrivia

- HW1 is assigned and due Sept 7, 12:30pm
- Office hours: Tu/Wed 1:45-2:30pm
- Also on zoom: `https://arizona.zoom.us/j/85228664366` password: CS573)
- Questionnaires due by August 31 **in person**
- Videos of lectures and lecture notes on D2L (today)
- Reading: Chapter 0, Chapter 1
- Last time: we introduced finite automata and regular languages
- Today: properties of regular languages, nondeterministic FAs, regular expressions, grammars

Recall: Finite Automaton



A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_s \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states.

Computation of a Finite Automaton

Let $M = (Q, \Sigma, \delta, q_S, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

- $r_0 = q_S$,
- $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, ..., n - 1$,
- $r_n \in F$.

A language is called a **regular language** if some finite automaton recognizes it.

Regular Operations

Let A and B be languages. We define the regular operations union, concatenation, and star as follows:

- Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$.
- Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$.
- Star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

A good idea to introduce:

- ϵ: the empty string
- ϵ: the empty language

Properties of Regular Languages

Theorem

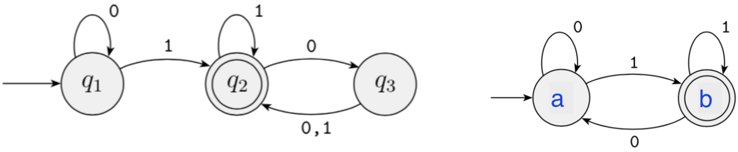
The class of regular languages is closed under the union operation.

Proof.

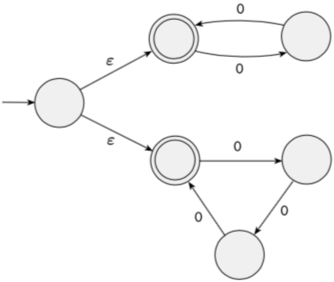
Given two regular languages A_1 and A_2 we want to show that $A_1 \cup A_2$ also is regular. Let M_1 be a finite automaton for A_1 and M_2 be a finite automaton for A_2. To prove that $A_1 \cup A_2$ is regular, we construct a finite automaton M that recognizes $A_1 \cup A_2$, using M_1 and M_2 as building blocks. M works by simultaneously simulating M_1 and M_2 and accepting if either of the simulations accept. M can keep track of the simulations by using as many states as the product of the states in M_1 and M_2. $\square$

Closure under Union

Example



Nondeterministic Finite Automata



A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_s, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma_\epsilon \to P(Q)$ is the transition function,
- $q_s \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states.

NFA Computation

Let $N = (Q, \Sigma, \delta, q_S, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of the alphabet Σ. Then N accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

- $r_0 = q_S$,
- $r_{i+1} \in \delta(r_i, w_{i+1})$, for $i = 0, ..., n-1$,
- $r_n \in F$.

An NFA accepts if any of its (possibly exponentially many) computation paths ends in an accept state.

Are Nondeterministic FA More Powerful?

What do we mean by *powerful*?

- the computational power of a machine is measured by the class of languages that it can recognize
- the computational power of a machine then is measured by the number of problems it can solve

Are Nondeterministic FA More Powerful?

What do we mean by *powerful*?

- the computational power of a machine is measured by the class of languages that it can recognize
- the computational power of a machine then is measured by the number of problems it can solve

So, while it's easier to design NFAs (they can be smaller and simpler), it turns out that NFAs recognize exactly the same class of languages as DFAs

NFAs

Theorem

A language is regular if and only if some NFA recognizes it.

Proof.

$\Rightarrow$ Let L be a regular language. Then by definition there exists a DFA for L. Since a DFA is a special case of an NFA (an NFA without epsilon transitions), we are done.

NFAs

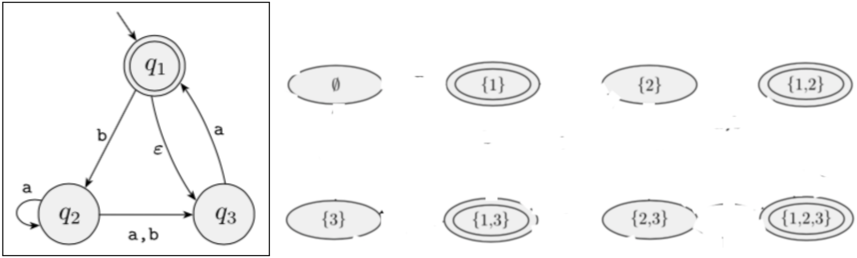
Theorem

A language is regular if and only if some NFA recognizes it.

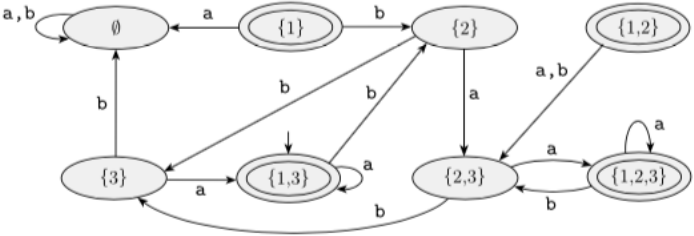
Proof.

$\Rightarrow$ Let L be a regular language. Then by definition there exists a DFA for L. Since a DFA is a special case of an NFA (an NFA without epsilon transitions), we are done.
$\Leftarrow$ Let N be an NFA. We will show how to create an equivalent DFA M. Then $L(N) = L(M)$ and by definition, this language is regular. $\square$

NFA to DFA Example



NFA to DFA Example



Equivalence of NFAs and DFAs (cont.)

Theorem

Every NFA has an equivalent DFA

Proof.

Let $N = (Q, \Sigma, \delta, q_s, F)$ be an NFA that recognizes language A. We will construct DFA $M = (Q', \Sigma, \delta', q'_s, F')$ that recognizes A.

- $Q' = P(Q)$
- for $R \in Q'$ and $a \in \Sigma$ let $\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a))$ for some $r \in R\}$
- $q'_S = E(q_s)$
- $F' = \{R \in Q' | R$ contains an accept state of $N\}$

where $E(R) = \{q | q$ can be reached from R by traveling along 0 or more ϵ arrows. □

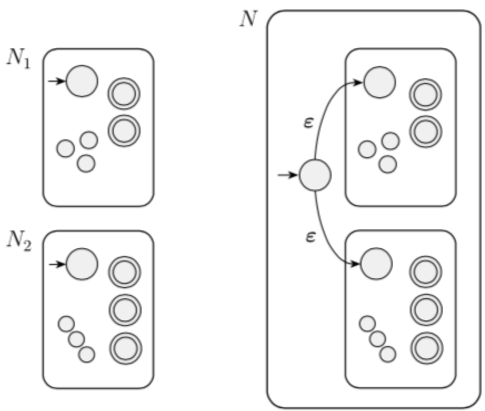
Closure under Union

Theorem

The class of regular languages is closed under the union operation.

Proof.

We proved this with DFAs; now we redo it with NFAs.



Closure under Union

Theorem

The class of regular languages is closed under the union operation.

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

- $Q = \{q_0\} \cup Q_1 \cup Q_2$.
- start state q_0
- $F = F_1 \cup F_2$.
- $$\delta(q, a) = \begin{cases} \delta_1(q, a) : q \in Q_1 \\ \delta_2(q, a) : q \in Q_2 \\ \{q_1, q_2\} : q = q_0, a = \epsilon \\ \emptyset : q = q_0, a \neq \epsilon \end{cases}$$

$\square$

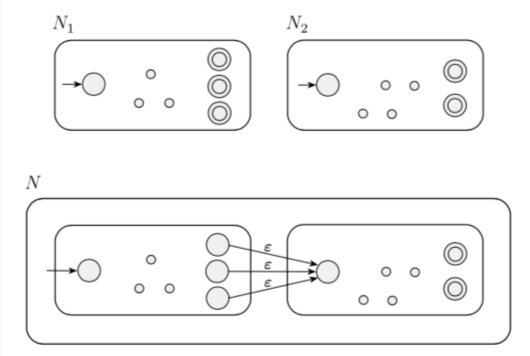
Closure under Concatenation

Recall concatenation: $A_1 \circ A_2 = \{xy | x \in A_1 \text{ and } y \in A_2\}$.

Theorem

The class of regular languages is closed under the concatenation operation.

Proof.



□

Closure under Concatenation

Theorem

The class of regular languages is closed under the concatenation operation.

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2. Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

- $Q = Q_1 \cup Q_2$.
- start state q_1
- accept states F_2
- $$\delta(q, a) = \begin{cases} \delta_1(q, a) : q \in Q_1, q \neq F_1 \\ \delta_1(q, a) : q \in F_1, a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\} : q \in F_1, a = \epsilon \\ \delta_2(q, a) : q \in Q_2 \end{cases}$$

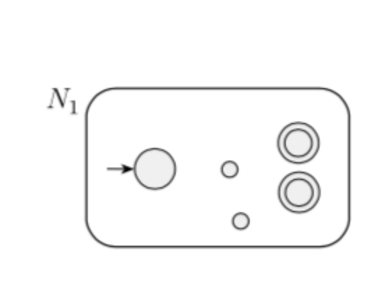
Closure under Star

Recall star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

Theorem

The class of regular languages is closed under the star operation.

Proof.



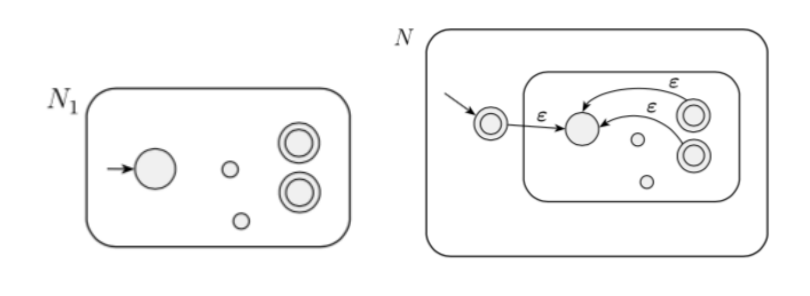
Closure under Star

Recall star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

Theorem

The class of regular languages is closed under the star operation.

Proof.



□

Closure under Star

Theorem

The class of regular languages is closed under the star operation.

Proof.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

- $Q = \{q_0\} \cup Q_1$
- start state q_0
- $F = \{q_0\} \cup F_1$.

$$\delta(q, a) = \begin{cases} \delta_1(q, a) : q \in Q_1, q \neq F_1 \\ \delta_1(q, a) : q \in F_1, a \neq \epsilon \\ \delta_1(q, a) \cup \{q_1\} : q \in F_1, a = \epsilon \\ \{q_1\} : q = q_0, a = \epsilon \\ \emptyset : q = q_0, a \neq \epsilon \end{cases}$$

□

Regular Expressions

R is a regular expression if R is

1. a for some a in the alphabet Σ,
2. ϵ,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

RE Examples

Let $\Sigma = \{0, 1\}$

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.
7. $01 \cup 10 = \{01, 10\}$.

RE $\Longleftrightarrow$ FAs

Theorem

A language is regular if and only if some regular expression describes it.

Proof.

$\Leftarrow$ Consider some regular expression R that describes language A. We show how to convert R into an NFA recognizing A.

1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$
2. $R = \epsilon$. Then $L(R) = \{\epsilon\}$, and the following NFA recognizes $L(R)$
3. $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$
4. $R = R_1 \cup R_2$
5. $R = R_1 \circ R_2$
6. $R = R_1^*$

RE $\iff$ FAs

Theorem

A language is regular if and only if some regular expression describes it.

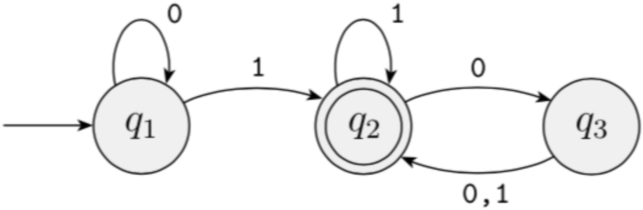
Proof.

$\Rightarrow$ Given some regular language A, there exists a DFA that recognizes A. We show how to extract an equivalent regular expression, by means a generalized nondeterministic finite automaton (GNFA). See textbook for complete proof. $\square$

Grammars

- We have seen two different, though equivalent, methods of describing languages: FAs and REs
- A grammar provides yet another way to describe a language
- While a FA *recognizes* a language, a grammar *generates* the language.
- We start with grammars for regular languages, also known as *type 3 grammars*

Type 3 Grammars: Example



$S \to 0S$
$S \to 1A$
$A \to 1A$
$A \to 0B$
$A \to \epsilon$
$B \to 0A$
$B \to 1A$

$S \to 0S|1A$
$A \to 1A|0B|\epsilon$
$B \to 0A|1A$

Type 3 Grammars

What is a grammar?

- substitution rules
- variables
- terminals
- start variable

$$S \to 0S|1A$$
$$A \to 1A|0B|\epsilon$$
$$B \to 0A|1A$$

What makes this a type 3 grammar?

- Chomsky's hierarchy: type 0, type 1, type 2, type 3
- the most general rule is $\alpha \to \beta$, where α and β are strings of terminals and variables
- rule restrictions: $\alpha \in V$ $|\alpha| = 1$, $|\beta| \leq 2$, etc.
- e.g., the grammar above it not only type 3 but also a right linear grammar