Closure properties

The closure properties tell us which operations let us stay within the class of regular languages, assuming we start from regular languages.

Theorem (Closure under regular operations)

If \( L_1, L_2 \) are regular, then so are:
- \( L_1 \cup L_2 \)
- \( L_1 \cdot L_2 \)
- \( L_1^* \)

Proof: since \( L_1, L_2 \) are regular, there are REs \( E_1, E_2 \) such that:
- \( \mathcal{L}(E_1) = L_1 \)
- \( \mathcal{L}(E_2) = L_2 \)

Then:
- \( L_1 \cup L_2 = \mathcal{L}(E_1) \cup \mathcal{L}(E_2) = \mathcal{L}(E_1 + E_2) \Rightarrow \) is regular
- \( L_1 \cdot L_2 = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2) = \mathcal{L}(E_1 \cdot E_2) \Rightarrow \) is regular
- \( L_1^* = (\mathcal{L}(E_1))^* = \mathcal{L}(E_1^*) \Rightarrow \) is regular

Q.E.D.

Closure under boolean operations:

If \( L_1 \) over \( \Sigma_1 \) and \( L_2 \) over \( \Sigma_2 \) are regular, then so are:
- \( L_1 \cup L_2 \) (union)
- \( \Sigma^* - L_1 \) (complement)
- \( L_1 \cap L_2 \) (intersection)

Note: to define the complement \( \overline{L} \) of a language \( L \), we need to specify the alphabet \( \Sigma \) of \( L \):
- \( \overline{L} = \Sigma^* - L \)

We may omit to specify \( \Sigma \) when it is clear from the context.
Theorem: (Closure under complementation)

If $L$ over $\Sigma$ is regular, then its complement is $\overline{L} = \Sigma^* - L$.

Proof:

Since $L$ is regular, there is a DFA

$$A_L = (Q, \Sigma, \delta, q_0, F) \text{ and } L(A_L) = L$$

Construct $\overline{A}_L = (Q, \Sigma, \delta, q_0, Q - F)$

Then $w \in L(\overline{A}_L)$ iff $\delta(q_0, w) \in Q - F$

iff $\delta(q_0, w) \notin F$

iff $w \notin L(A_L)$

Hence $L(\overline{A}_L) = L(A_L) = \overline{L}$, and $\overline{L}$ is regular. q.e.d.

Note: In order to obtain the complement by complementing the set of final states, the automaton has to be deterministic.

Example: let $A_0$ be the NFA

\[ \begin{array}{c}
q_0 \xrightarrow{0, 1} q_1 \\
q_1 \xrightarrow{0, 1} q_0 \\
\end{array} \]

$L(A_0) = \{ w | w \text{ ends with } e 0 \}$

If we take $A'_0$ with

\[ \begin{array}{c}
q_0 \xrightarrow{0, 1} q_1 \\
q_1 \xrightarrow{0, 1} q_0 \\
\end{array} \]

then $L(A'_0) = \Sigma^* = L(A_0)$

Hence, in general, given an NFA $A_N$, to obtain an automaton for $L(A_N)$ we first have to determinize $A_N$ (e.g., by applying the subset construction). => Sequentiality is lost.

Exercise E4.1: By referring to examples we have seen, prove that in general we cannot do better to compute a DFA for the complement of a language accepted by an NFA.
Theorem (closure under intersection)

If $L_1, L_2$ are regular, then so is $L_1 \cap L_2$

**Proof:** we simply use De Morgan's law

$$L_1 \cap L_2 = \overline{L_1 \cup L_2}$$

and exploit closure under $\cap$ and $\cup$.

**Note:** this proof is constructive, i.e. given e.g. NFA's $A_1, A_2$ for $L_1$ and $L_2$, it tells us how to construct an NFA for $L_1 \cap L_2$.

What is the cost of this construction? Exponential.

In fact, there is a direct construction that computes, given two NFA's $A_1, A_2$, an NFA $A_{1 \cap 2}$ for $L(A_1) \cap L(A_2)$.

If $A_1$ and $A_2$ have respectively $n_1$ and $n_2$ states, then $A_{1 \cap 2}$ has $n_1 n_2$ states. ($A_{1 \cap 2}$ is called product automaton.)

See book for details. [Exercise]

---

**Closure under reversal**

**Definition:**

- reversal of a string: $\varepsilon^R = \varepsilon$

- if $w = a_1 \ldots a_n$ then $w^R = (a_1 \ldots a_n)^R = a_n \ldots a_1$

- reversal of a language: $L^R = \{ w^R \mid w \in L \}$

---

**EXERCISE**
Theorem (closure under reversal)

If \( L \) is regular, then so is \( L^R \)

Proof: we extend reverse to R.E., inductively

Base: \( E^R = \varepsilon \)
\( \varepsilon^R = \varepsilon \)
\( \phi^R = \phi \)
\( a^R = a \) for \( a \in \Sigma \)

Induction:
\( (E_1 + E_2)^R = E_1^R + E_2^R \)
\( (E_1 \cdot E_2)^R = E_2^R \cdot E_1^R \)
\( (E_1^*)^R = (E_1^R)^* \)

We prove by structural induction that \( \mathcal{J}(E^R) = (\mathcal{J}(E))^R \)

Base: clear

Induction:
\( \mathcal{L}((E_1 + E_2)^R) = \) \( \mathcal{L}(E_1^R + E_2^R) = \) \([\text{Def. of reversal for R.E.}] \)
\( = \mathcal{L}(E_1^R) \cup \mathcal{L}(E_2^R) = \) \([\text{Semantics of +}] \)
\( = (\mathcal{J}(E_1))^R \cup (\mathcal{J}(E_2))^R = \)
\( = (\mathcal{J}(E_1))^R \cup (\mathcal{J}(E_2))^R = \) \([\text{I.H.}] \)
\( = (\mathcal{J}(E_1))^R \cup (\mathcal{J}(E_2))^R = \)
\( = \{w^R \mid w \in \mathcal{J}(E_1)\} \cup \{w^R \mid w \in \mathcal{J}(E_2)\} = \)
\( = \{w^R \mid w \in \mathcal{J}(E_1) \cup \mathcal{J}(E_2)\} = \)
\( = (\mathcal{J}(E_1 \cup E_2))^R = \) \([\text{Semantics of +}] \)
\( = (\mathcal{J}(E_1 + E_2))^R \)

Other cases: exercise

Example: \( E = a \cdot b \cdot c + b \cdot c^* \cdot a \)
\( E^R = c \cdot b \cdot e^* \cdot c \)
\( \uparrow \text{EXERCISE} \)
Consider: \( L_{\text{alt}} = \{ w \mid \text{has alternating 0's and 1's} \} \)
\[ L_{\text{eq}} = \{ w \mid \text{has an equal number of 0's and 1's} \} \]

Claim: \( L_{\text{alt}} \) is regular

Proof: every \( E_{\text{alt}} = (3+0)(1.0)^* (3+1) \) is such that \( L(E_{\text{alt}}) = L_{\text{alt}} \)

Claim: \( L_{\text{eq}} \) is not regular

How can we prove this?

Intuition: DFA with \( n \) states can count up to \( n \).

To decide whether \( w \in L_{\text{eq}} \), we need unbounded counting (since we may be arbitrarily long)

Pumping Lemma:

For all regular languages \( L \subseteq \Sigma^* \), there exists \( n \) (which depends on \( L \)) such that for all \( w \in L \) with \( |w| \geq n \), there exists a decomposition \( w = xyz \) of \( w \) s.t.

1) \( |y| \geq 1 \) (i.e., \( y \neq \varepsilon \))
2) \( |xy| \leq n \)
3) for all \( k \geq 0 \), \( xy^k z \in L \).

Intuitively, for every \( w \in L \), we can find a substring \( y \) "near" the beginning of \( w \) that can be "pumped", while still obtaining words in \( L \).
Given regular language $L$, let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L(A) = L$.

We take $m = |Q|$. 

Consider any $w = e_1 e_2 \ldots e_m \in L$ with $m = |w| \geq m$.

Since $w \in L(A)$, we have that $\hat{\delta}(q_0, w) \in F$.

Define $\pi_i = \hat{\delta}(q_0, e_1 e_2 \ldots e_i) \quad \forall i \in \{1, \ldots, m\}$ and 

$\pi_0 = q_0$.

Since $m \geq m$,

- each $\pi_i$, $0 \leq i \leq m$, belongs to $Q$, and
- $|Q| = m$.

By the pigeon-hole principle, $\pi_0, \pi_1, \ldots, \pi_m$ are not all distinct.

Let $i, j$ with $0 \leq i < j \leq m$ be the least indices such that

$\pi_i = \pi_j$.

Hence, to accept $w$, the DFA goes through a cycle:

\[
\begin{array}{c}
\pi_0 \xrightarrow{e_1} \pi_1 \xrightarrow{e_2} \pi_2 \xrightarrow{\cdots} \pi_m \xrightarrow{e_m} \pi_0
\end{array}
\]

Observe:
- $|j-i| = j-i > 1$ (since $i < j$)
- $1 \leq j-i \leq m$

\[
\hat{\delta}(q_0, y^j i^2) = \hat{\delta}(\delta(q_0, y^j), y^i) = \hat{\delta}(\hat{\delta}(q_0, x^j), y^i) = \hat{\delta}(q_m, y^i) = \pi_m 
\]

\[
\rightarrow \cdots = \hat{\delta}(\hat{\delta}(q_0, y^j), y^2) = \cdots = \hat{\delta}(q_0, y^j) = \pi_0 \in F \Rightarrow x y^i z \in L.
\]
The pumping lemma states a property of R.L. that can be used to show that a given language is not regular.

Idea: pick \( w \in L \) such that we can easily show that \( xy^kz \in L \) for some choice of \( k \).

Difficulty: we must do so regardless of the choices for \( n \), and the decomposition \( x, y, z \).

More precisely: to show that \( L \) is not regular, we have to show that:

For all \( n \),

there exists a \( w \in L \) with \( |w| > n \) such that

for all decompositions \( w = xy^2z \) of \( w \)...

with \( |y| < n \)

\( |x.y| \leq n \)

there exists \( k > 0 \) s.t. \( x.y^k.z \notin L \)

We can view the alternation of \( A \) and \( E \) as a game between Alice and Ed:

- Ed chooses the language \( L \) he wants to show nonregular.
- Alice chooses \( n \).
- Ed chooses \( w \in L \) with \( |w| > n \).
- Alice chooses a decomposition \( w = xy^2z \) with \( |y| < n \)

- Ed chooses \( k > 0 \), and he wins iff \( x.y^k.z \notin L \).

Then \( L \) is not regular if Ed has a winning strategy, i.e., he can win whatever moves Alice makes (respecting the rules).
Example: Let \( L_{eq} \) be not regular.

Let's play the game and show that \( Ed \) can always win.

- \( Ed \) chooses \( L_{eq} \)
- Alice chooses some \( m \)
- \( Ed \) chooses \( w = 0^m 1^m \)
  
  note that \( w \in L \) and \( |w| \geq m \)
- Alice chooses a decomposition \( w = x \cdot y \cdot z \)
  
  with \( y \neq \epsilon \) and \( |x \cdot y| \leq m \)
  
  note that, since \( |x \cdot y| \leq m \), we have \( x \cdot y = 0 \cdots 0 \)
  
  \( \Rightarrow \) let \( k = 0 \cdots 0 \), \( y = 0 \cdots 0 \)
  
  \( a \), \( b \geq 1 \)
  
  then \( w = 0^k \cdot 0^b \cdot 0^{m-a-b} 1^m \)
  
  \( k = y \), \( z = \epsilon \)

- \( Ed \) chooses \( k = 0 \)
  
  then \( x \cdot y \cdot z = k \cdot z = 0^k \cdot 0^{m-a-b} 1^m = 0^{m-b} 1^m \notin L \)
  
  and \( Ed \) wins

\( \Rightarrow \) \( L_{eq} \) is not regular.

Exercise: E4.2 Let \( L_{prime} = \{ w \in \{0\}^* \mid |w| \text{ is prime} \} \).

Show that \( L_{prime} \) is not regular.

Notice that the converse of the pumping lemma does not hold.

In terms of the game between Alice and Ed,

- \( L \) is not regular \( \iff \) \( Ed \) has a winning strategy

Example: consider \( L = L_1 \cdot L_2 \) with \( L_1 \) regular

We know that \( L \) is not regular

but \( Ed \) does not have a winning strategy
Decision problems for regular languages

Decision problem: Let $\mathcal{O}$ be some property of languages.

Input: Regular language $L$. (represented as DFA, NFA, $\varepsilon$-NFA, or RE.)

Output: does $L$ have property $\mathcal{O}$? < yes, no

A decision algorithm decides a decision problem:
- means: - correct answer
- always terminates in finite time

Emptiness: decide if a regular language $L$ is empty.

When $L$ is given as an automaton, then $L$ is not empty iff a final state is reachable from the initial state.

This is an instance of graph reachability: recursively
- base: the initial state is reachable
- induction: if $q$ is reachable, and $\delta(q, a) = p$ for some $a$, then $p$ is reachable.

For $n$ states, this takes at most $O(n^2)$
(actually, it takes at most the number of arcs)

Exercise: Emptiness, when $L$ is given as a RE.

Let us compute $\text{empty}(E)$ by structural induction on $E$

- base: $\text{empty}(\emptyset) = \text{true}$
- $\text{empty}(-E) = \text{false}$
- $\text{empty}(E) = \text{false}$ $\forall a \in \Sigma$

- induction: $\text{empty}(E^*) = \text{false}$
  - $\text{empty}(E \cdot E_1) = \text{false}$
  - $\text{empty}(E_1 \cdot E_2) = \text{empty}(E_1) \land \text{empty}(E_2)$
  - $\text{empty}(E_1 \cdot E_2) = \text{empty}(E_1) \lor \text{empty}(E_2)$

$\Rightarrow$ hence in $E$
Membership: given \( w \in \Sigma^* \) and \( L \subseteq \Sigma^* \), with \( L \) regular, decide whether \( w \in L \).

Algorithm:
- when \( L \) is given as a DFA \( A_D \):
  - simulate the run of \( A_D \) on \( w \)
  - if transition table is stored as a 2-dimensional array, each transition takes constant time
    \( \Rightarrow \) test takes linear time in \( |w| \)
- when \( L \) is given as an NFA \( A_N \):
  - if we compute the equivalent DFA \( \Rightarrow \) exponential in \( |A_N| \), linear in \( |w| \)
  - we can also simulate directly the NFA, by computing the sets of states the NFA is in after each input symbol
    \( \Rightarrow O(|w| \cdot \Sigma^2) \) where \( \Sigma \) is the number of states of \( A_N \)
      at each step at most \( \Sigma \) states
      each with at most \( \Sigma \) successors

Equality: given regular languages \( L_1, L_2 \), decide whether \( L_1 = L_2 \)

Idea: reduce to emptiness:

Consider \( L = (L_1 \cap L_2) \cup (E_1 \cap L_2) \) (symmetric difference)

\( L \) is regular, by closure of \( \cap, \cup, \neg \)

then \( L_1 = L_2 \ \Leftrightarrow \ L = \emptyset \)

Algorithm: 1) Compute representation for \( L \) (as DFA on \( R.B. \))
  2) Decide emptiness of \( L \)
Finiteness: given regular language \( L \)

decide whether \( L \) is finite.

Let \( A_L \) be a DFA for \( L \) with \( n \) states.

**Theorem:** \( L \) is infinite iff \( \exists w \in L \) s.t. \( n \leq |w| < 2n \).

**Proof:** \( \Leftarrow \) Let \( w \in L \) with \( n \leq |w| \).

By pumping lemma, \( w = x \cdot y \cdot z \) with \( y \neq \varepsilon \)
and \( \forall k \geq 0, x \cdot y^k \cdot z \in L \).

Hence \( L \) is infinite.

\( \Rightarrow \) Suppose \( L \) is infinite.

Then \( \exists w \in L \) s.t. \( |w| \geq n \) (there are only finitely many strings of length \( < n \)).

Set \( \tilde{w} \) be the shortest string in \( L \) of length \( \geq n \).

Claim: \( |	ilde{w}| < 2n \)

Proof by contradiction: suppose \( |\tilde{w}| \geq 2n \)

By pumping lemma, \( \tilde{w} = x \cdot y \cdot z \) with \( |x \cdot y| \leq n \)

and \( x \cdot y^k \cdot z = x \cdot z \in L \)

We have:

1) \( |x \cdot z| = |\tilde{w}| - |y| \geq 2n - n = n \)

2) \( |x \cdot z| < |\tilde{w}| \), since \( |y| \geq 1 \)

This contradicts choice of \( \tilde{w} \) as shortest string,

which proves the claim.

Hence, we have a string \( \tilde{w} \in L \) with \( n \leq |\tilde{w}| < 2n \)

q.e.d.
From the theorem we get an algorithm for finiteness.

Algorithm: For each \( w \in \Sigma^* \) with \( n \leq |w| < 2n \),

test whether \( w \in L \).

Exercise 4.3.3. Give an algorithm to decide whether a regular language \( L \) is minimal, i.e., \( L = \Sigma^* \).

Exercise 4.3.4. Give an algorithm to decide whether two regular languages \( L_1 \) and \( L_2 \) have at least one string in common.

Exercise E 4.3. Give an algorithm to decide whether a regular language \( L_1 \) is contained in another regular language \( L_2 \).
State minimization.

Given DFA \( A = (Q, \Sigma, \delta, q_0, F) \), find \( A' \) with minimal
number of states s.t. \( L(A') = L(A) \).

Idea: partition \( Q \) into equivalence classes and collapse equivalent
states.

Equivalence relation on states:

\[ p \equiv q \text{ if for all } w \in \Sigma^*: \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \]

The equivalence relation induces a partition of \( Q \)
\[ Q = C_1 \cup C_2 \cup \ldots \cup C_k \]
for all \( p \in C_i, q \in C_j \): \( p \equiv q \iff i = j \)

How do we find the partition? We discover inequivalent states:

\[ p \not\equiv q \text{ if for some } w \in \Sigma^* \hat{\delta}(p, w) \in F \text{ and } \hat{\delta}(q, w) \notin F \]
on vice versa.

Let \( w = e_1 e_2 \ldots e_m \) (i.e. \( |w| = m \))

\[ p \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \longrightarrow \ldots \xrightarrow{e_{m-1}} q_{m-1} \xrightarrow{e_m} q_m \]

\[ q \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \longrightarrow \ldots \xrightarrow{e_{m-1}} q_{m-1} \xrightarrow{e_m} q_m \]

One is final and the other is not.

Note: \( e_1 e_2 \ldots e_m \) is a proof of length \( m-i \) of inequivalence
of \( p_i \) and \( q_i \).

Definition: \( p \equiv_i q \) if for all \( w \) with \( |w| \leq i \)
\[ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \]

(intuitively, there is no inequivalence proof of length \( \leq i \)).
The following is immediate to see:
\[ p \neq i + q \] if and only if for some \( c \in \Sigma \)
\[ \delta(p, c) \neq \delta(q, c). \]

Algorithm to compute \( \equiv_i \) inductively on \( i \):

Step 0: partition \( Q = C_1 \cup C_2 \) with \( C_1 = F, \ C_2 = Q - F \)
justified since \( p \neq q \) iff one is final and
the other not.

Step i+1: determine \( p \equiv_i q \) iff \( \forall c \in \Sigma \)
\[ \delta(p, c) \equiv_i \delta(q, c) \]
compute refined partition.

Algorithm terminates when the refined partition coincides with
the one in the previous step (at most \(|Q| \) steps).

Example:

Step 0: \( C_1^0 = \{1, 2, 5, 6\} \quad \quad \quad C_2^0 = \{3, 4\} \)

Step 1: \( C_1^1 = \{1, 2, 5, 6\} \quad \quad \quad C_2^1 = \{3\} \quad \quad \quad C_3^1 = \{4\} \)

Step 2: \( C_1^2 = \{1, 5, 6\} \quad \quad \quad C_2^2 = \{2\} \quad \quad \quad C_3^2 = \{3\} \quad \quad \quad C_4^2 = \{4\} \)

Step 3: \( C_1^3 = \{1\} \quad \quad \quad C_2^3 = \{2\} \quad \quad \quad C_3^3 = \{3\} \quad \quad \quad C_4^3 = \{4\} \quad \quad \quad C_5^3 = \{5, 6\} \)

Step 4: no change

10/12/2007

8/11/2009
To construct $A'$:

1. Construct partition $Q = C_1 \cup \ldots \cup C_n$ of states of $A$.
2. Construct $A' = (Q', \Sigma, \delta', q'_0, F')$
   
   - States $Q' = \{ C_1, C_2, \ldots, C_n \}$
   
   - Transitions: if $\delta(q, a) = q$ in $A$
     
     then $\delta(C[q], a) = C[q]$

     where $C[q]$ is the equivalence class of $q$

   - Start state: $C[q_0]$
   
   - Final states: $\{ C[q_6] \mid q_6 \in F \}$

We can verify that $A'$ is a well-defined DFA.

**Example:**

![Diagram](image)

Note that $C_5$ is not reachable from the start state and must be removed.

We could show that the DFA constructed in this way is the smallest possible for a given language.

**Myhill-Nerode Theorem:**

Given $L \subseteq \Sigma^*$, consider the equivalence relation $R_L$ on $\Sigma^*$

defined as follows: $x R_L y \iff \forall z \in \Sigma^*: xz \in L \iff yz \in L$.

Then $L$ is regular iff $R_L$ induces a finite number of equivalence classes.