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:

\[
L_1 \cup L_2 \quad L_1^* \quad L_2^*
\]

**Proof:** since \( L_1, L_2 \) are regular, there are R.E.s \( E_1, E_2 \) s.t.

\[
\mathcal{L}(E_1) = L_1 \quad \mathcal{L}(E_2) = L_2
\]

Then:

\[
L_1 \cup L_2 = \mathcal{L}(E_1) \cup \mathcal{L}(E_2) = \mathcal{L}(E_1 \cup E_2) \quad \Rightarrow \text{is regular}
\]

\[
L_1^* = \mathcal{L}(E_1)^* = \mathcal{L}(E_1^*) \quad \Rightarrow \text{is regular}
\]

\[
L_2^* = \mathcal{L}(E_2)^* = \mathcal{L}(E_2^*) \quad \Rightarrow \text{is regular}
\]

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 $L$ is regular, then $\overline{L}$ is

Proof:

Since $L$ is regular, there is a DFA $A_L$, i.e., $L(A_L) = L$.

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

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

iff $\hat{\delta}(q_0, w) \notin F$

iff $w \notin L(A_L)$

Hence $L(A_{\overline{L}}) = \overline{L(A_L)} = \overline{L}$, 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

![NFA Diagram]

$L(A_0) = \{w \mid w \text{ ends with } \epsilon \}$

If we take $A_0'$ with

![NFA Diagram]

then $L(A_0') = \Sigma^* \neq \overline{L(A_0)}$

Hence, in general, given an NFA $A_N$, to obtain an automaton for $\overline{L(A_N)}$ we first have to determinize $A_N$ (e.g., by applying the subset construction), => sequential blowup.

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 the 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

\[
L_1 \cap L_2 = L_1 \setminus \overline{L_1} \cup L_2
\]

and exploit closure under \( \setminus \) and \( \cup \).

**Note:** this proof is constructive, i.e., given e.g., NFA’s \( \mathcal{N}_1 \) for \( L_1 \) and \( \mathcal{N}_2 \), 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, i.e.,

given two NFA’s \( \mathcal{A}_1, \mathcal{A}_2 \), an NFA \( \mathcal{A}_{1 \cap 2} \) for \( \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \).

If \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) have respectively \( m_1 \) and \( m_2 \) states, then \( \mathcal{A}_{1 \cap 2} \) has \( m_1 \cdot m_2 \) states. (\( \mathcal{A}_{1 \cap 2} \) is called product automaton)

See book for details.

\[ \downarrow \text{EXERCISE} \]

---

**Closure under reversal.**

**Definition:**

- **reversal of a string:**
  
  \[ \varepsilon^R = \varepsilon \]

  \[ (\varepsilon_1 \ldots \varepsilon_n)^R = \varepsilon_n \ldots \varepsilon_1 \]

- **reversal of a language:**
  
  \[ L^R = \{ w^R \mid w \in L \} \]
Theorem (closure under reversal)

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

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

base: \( \varepsilon^R = \varepsilon \)
\( \emptyset^R = \emptyset \)
\( a^R = a \) \( \text{for } a \in \Sigma \)

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 \( \mathcal{L}(E^R) = (\mathcal{L}(E))^R \)

base: clear

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

Other cases: exercise

Example: \( E = aE + \varepsilon \)
\( \Rightarrow (E^R = aE + \varepsilon^R)^R = a^R E^R + \varepsilon^R \)

↑ 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: easy \( E_{\text{alt}} = (3+0)(1+0)^* (3+1) \) is such that \( S(F_{\text{alt}}) = L_{\text{alt}} \)

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

How can we prove this?

Definition: DFA with \( m \) states can count up to \( m \).

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

Pumping Lemma:

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

1) \( |y| \geq 1 \) (i.e., \( y \neq \epsilon \))
2) \( |xy| \leq m \)
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 \( n = |Q| \).

Consider any \( w = a_1 a_2 \cdots a_m \in L \) with \( m = |w| \geq n \).

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

Define \( \hat{q}_i = \hat{\delta}(q_0, a_1 a_2 \cdots a_i) \) \( \forall i \in \{1, \ldots, m\} \) and \( \hat{q}_0 = q_0 \).

Since \( m \geq n \),

- each \( \hat{q}_i \), \( 0 \leq i \leq m \) belongs to \( Q \), and
- \( |Q| = m \)

by the pigeon-hole principle, \( \hat{q}_0, \hat{q}_1, \ldots, \hat{q}_m \) are not all distinct.

Let \( i, j \) with \( 0 \leq i < j \leq m \) be the least indices such that \( \hat{q}_i = \hat{q}_j \).

Hence, to accept \( w \), the DFA goes through a cycle.

\[ \hat{\delta}(\hat{q}_i, \hat{a}_1) = \hat{q}_i \]

\[ \hat{\delta}(\hat{q}_i, \hat{a}_2) = \hat{q}_i \]

\[ \cdots \]

\[ \hat{\delta}(\hat{q}_i, \hat{a}_m) = \hat{q}_m \]

Observe: \( |\hat{y}| = j-i \geq 1 \) (since \( i < j \))

- \( 1 \times |\hat{y}| = j \leq m \)

\[ \hat{\delta}(q_0, \hat{a}_1 \hat{a}_2) = \hat{\delta}(\hat{\delta}(q_0, \hat{a}_1), \hat{a}_2) = \hat{\delta}(\hat{\delta}(\hat{q}_i, \hat{a}_1), \hat{a}_2) = \hat{\delta}(\hat{\delta}(q_0, \hat{a}_1), \hat{a}_2) = \hat{\delta}(\hat{q}_i, \hat{a}_2) = \hat{q}_i \]

\[ \hat{\delta}(q_0, \hat{a}_1 \hat{a}_2 \cdots \hat{a}_j) = \hat{\delta}(\hat{\delta}(q_0, \hat{a}_1 \hat{a}_2 \cdots \hat{a}_{j-1}), \hat{a}_j) = \cdots = \hat{\delta}(\hat{\delta}(q_0, \hat{a}_1), \hat{a}_j) = \hat{\delta}(q_0, \hat{a}_1) = \hat{q}_0 \in F \Rightarrow \hat{a}_1 \hat{a}_2 \cdots \hat{a}_j \in L \]
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 \( x y^k z \in L \) 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:

\[
\text{for all } n
\]

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

\[
\text{for all decompositions } w = x y z \text{ of } w \text{ with } |y| \leq 1,
\]
\[
|x y| \leq n
\]

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 non-regular.
- Alice chooses \( n \).
- Ed chooses \( w \in L \) with \( |w| > n \).
- Alice chooses a decomposition \( w = x y z \) with \( |y| \leq 1 \).
- 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 (regesting the rules).
Example: Let $L$ be not regular

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

- Ed chooses $L$
- Alice chooses some $m$
- Ed chooses $w = 0^m 1^m$
  
  note that $w \in L$ and $|w| > m$
- Alice chooses a decomposition $w = x \cdot y \cdot z$
  
  with $y \neq \emptyset$ and $|x \cdot y| \leq m$
  
  note that, since $|x \cdot y| \leq m$, we have $x \cdot y = 0 \cdots 0$
  
  $\rightarrow$ let $x = \overbrace{0 \cdots 0}^{a}$ $y = \overbrace{0 \cdots 0}^{b}$ $z = \overbrace{0 \cdots 0}^{c}$

  then $w = \underbrace{0 \cdots 0}_x \cdot \overbrace{0^m \cdots 0^m}_{y} \cdot \overbrace{0 \cdots 0}^z$

- Ed chooses $k = 0$

  then $x \cdot y \cdot z = xz = \overbrace{0 \cdots 0}^{a} \cdot \overbrace{0^m \cdots 0^m \cdots 0^m}^{m-k} \cdot \overbrace{0 \cdots 0}^{1} = \overbrace{0 \cdots 0}^{m-k} \cdot 1 \notin L$

  and Ed wins.

$\rightarrow$ $L$ is not regular.

Exercise: E4.2

Let $L_{\text{prime}} = \{ w \in \{0\}^* \mid |w| \text{ is prime} \}$.

Show that $L_{\text{prime}}$ 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 $L_2$ not regular.

We have that $L$ is not regular.

But Ed does not have a winning strategy.
Decision problems for regular languages

Decision problem: Set G be some property of languages

Input: regular language L (represented as DFA, NFA, E-NFA, or R.E.)

Output: does L have property \( G \) (< yes

A decision algorithm decides a decision problem:

\[ \text{means: correct answer} \]
\[ \text{always terminates in finite time} \]

\[ \text{Emptyness: decide if a regular language } L \text{ is empty} \]

When \( L \) is given as an automaton, then \( L \) is not empty
if 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 \( O(n^2) \)

(exactly, it takes at most the number of arcs)

Exercise: Emptyness, when \( L \) is given as a R.E.

Let us compute empty \( (E) \) by structural induction on \( E \)

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

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

\( \Rightarrow \text{linear in } E \)
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
  $\Rightarrow O(|w| \cdot n^2)$ where $n$ is the number of states of $A_N$
  - at each step at most $n$ states, each with at most $n$ successors

Equivalence: given regular languages $L_1, L_2$, decide whether $L_1 = L_2$.

Idea: reduce to emptiness:
- consider $L = (L_1 \cap \overline{L_2}) \cup (L_1 \cap \overline{L_2})$ (symmetric difference)
  - $L$ is regular, by closure of $\cap, \cup$,
  - then $L_1 = L_2 \iff L = \emptyset$

Algorithm:
1) Compute representation for $L$ (as DFA or R.E.)
2) Decide emptiness of $L$
Liveness: given regular language $L$

decide whether $L$ is finite.

Let $A_L$ be a DFA for $L$ with $m$ states.

**Theorem:** $L$ is infinite iff $\exists w \in L$ s.t. $m < |w| < 2m$.

**Proof:** 

Let $w \in L$ with $m < |w| < 2m$.

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. $|w| \geq m$ (there are only finitely many strings of length $< m$)

Set $\bar{w}$ be the shortest string in $L$ of length $\geq m$.

**Claim:** $|\bar{w}| < 2m$

**Proof by contradiction:** Suppose $|\bar{w}| \geq 2m$

By pumping lemma, $\bar{w} = x \cdot y \cdot z$ with $|x \cdot y \cdot z| \leq m$

and $x \cdot y \cdot z = x \cdot z \in L$

We have:

1) $|x \cdot z| = |\bar{w}| - |y| \geq 2m - m = m$

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

This contradicts choice of $\bar{w}$ as shortest string,

which proves the claim.

Hence, we have a string $\bar{w} \in L$ with $m < |\bar{w}| < 2m$

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

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

**Exercise E4.3** Provide 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 minimum number of states s.t. $L(A') = L(A)$.

Idea: partition $Q$ into equivalence classes and collapse equivalent states.

Equivalence relation on states:

$p \equiv q$ 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 $Q$

$Q = C_1 \cup C_2 \cup \cdots \cup C_k$

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:

$p \not\equiv q$ if for some $w \in \Sigma^*$: $\hat{\delta}(p, w) \in F$ and $\hat{\delta}(q, w) \notin F$ or vice versa.

Let $w = e_1 e_2 \cdots e_m$, (i.e. $|w| = m$)

$p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \rightarrow \cdots \rightarrow p_{m-1} \xrightarrow{e_m} p_m \leftarrow$ one is final and

$q \xrightarrow{e_1} q_1 \xrightarrow{a_2} q_2 \rightarrow \cdots \rightarrow q_{m-1} \xrightarrow{e_m} q_m \rightarrow$ the other is not.

Note: $e_1 e_2 \cdots e_m$ 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:

\[ q \neq \pi_{n+1} q \text{ if and only if for some } e \in \Sigma \]

\[ \delta(q, e) \neq \pi \delta(q', e). \]

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

Step 0: partition \( Q = C_1 \cup C_2 \) with \( C_1 = F \), \( C_2 = Q - F \)

(justified since \( q \neq q' \) iff one is final and the other not)

Step \( i+1 \): determine \( q = \pi_{n+1} q \) iff \( \forall e \in \Sigma \)

\[ \delta(q, e) \in i \delta(q', e) \]

compute refined partition

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

**Example:**

\[
\begin{array}{c}
\text{Step 0: } C_1 = \{4, 5, 6\} & C_2 = \{3, 4\} \\
\text{Step 1: } C_1 = \{4, 5, 6\} & C_2 = \{3\} & C_3 = \{4\} \\
\text{Step 2: } C_1 = \{4, 5, 6\} & C_2 = \{2\} & C_3 = \{3\} & C_4 = \{4\} \\
\text{Step 3: } C_1 = \{4\} & C_2 = \{2\} & C_3 = \{3\} & C_4 = \{4\} & C_5 = \{5, 6\} \\
\text{Step 4: no change}
\end{array}
\]

\( \text{8/11/2004} \)
To construct $A'$:

1) Construct partition $Q = C_1 \cup \ldots \cup C_k$ of states of $A$

2) Construct $A' = (Q', \Sigma, \delta', q_0', F')$

- States $Q'$ = \{ $C_1, C_2, \ldots, C_k$ \}
- 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: \{ $C[q_\#] \mid q_\# \in F$ \}

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

Exercise E.4.4

**Example:**

![Diagram of DFA]

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.

Mysfill - Nerode Theorem:

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