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 $E_n$, $E_2$ such that $L(E_n) = L_1$ and $L(E_2) = L_2$.

Then:

- $L_1 \cup L_2 = L(E_n) \cup L(E_2) = L(E_1 \cup E_2) \Rightarrow$ in regular
- $L_1 \cdot L_2 = L(E_n) \cdot L(E_2) = L(E_1 \cdot E_2) \Rightarrow$ in regular
- $L_1^* = (L(E_n))^* = L(E_1^*) \Rightarrow$ in 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 $\overline{L} = \Sigma^* - L$.

Proof:
Since $L$ is regular, there is a DFA

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

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

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

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

iff $w \notin L(A_L)$

Hence $L(A_\overline{L}) = \overline{L}$, and $\overline{L}$ is regular. Q.E.D.

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

Example: let $A_0$ be the NFA

$$
\begin{array}{c}
\circlearrowleft \\
q_0 \quad 0, 1 \\
\rightarrow q_1
\end{array}
$$

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

If we take $A_0'$ with

$$
\begin{array}{c}
\circlearrowleft \\
q_0 \quad 0, 1 \\
\rightarrow q_1
\end{array}
$$

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

Hence, in general, given an NFA $A_N$, to obtain a DFA for $\overline{L(A_N)}$ we first have to determinize $A_N$ (e.g., by applying the subset construction), \Rightarrow exponential 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

$$L_1 \cap L_2 = L_1 \cup L_2$$

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

Note: this proof is constructive, i.e. given e.g. NFA $A$ 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 $m_1$ and $m_2$ states, then $A_{1 \cap 2}$ has $m_1 \cdot m_2$ 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 $w = e_1 \cdots e_m$, then $w^R = (e_m \cdots e_1)^R = e_1 \cdots e_m$

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

\[ \downarrow \text{EXERCISE} \]
Theorem (closure under reversal)

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

**Proof:** We extend reversal to R.E., inductively

**Base:** \( E^R = E \)

\[ \emptyset^R = \emptyset \]

\[ a^R = a \text{ for } a \in \Sigma \]

**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 \( J(E^R) = (J(E))^R \)

**Base:** clear

**Induction:**

\[ J((E_1 + E_2)^R) = J(E_1^R + E_2^R) \]

[Def. of reversal for R.E.]

\[ = J(L(E_1^R) + L(E_2^R)) = \]

[Semantics of +]

\[ = J(L(E_1^R)) \cup J(L(E_2^R)) = \]

[S.H.]

\[ = (J(E_1))^R \cup (J(E_2))^R = \]

\[ = \{ w^R \mid w \in J(E_1) \} \cup \{ w^R \mid w \in J(E_2) \} = \]

\[ = \{ w^R \mid w \in J(E_1) \cup J(E_2) \} = \]

\[ = (J(E_1) \cup J(E_2))^R = \]

[Semantics of +]

\[ = (J(E_1 + E_2))^R \]

Other cases: 

**Example:** 

\[ E = a.b.c + b.c^*a \]

\[ E^R = c^*b.e + a.c^*b \]

↑ **EXERCISE**
Proving languages not to be regular

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 \( \phi(E_{\text{alt}}) = L_{\text{alt}} \)

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 \( w \) 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 \( |w| > n \)
there exists a decomposition \( w = xyz \) of \( w \) s.t.

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

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

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

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

Since \( m \geq n \),

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

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

Let \( i, j, \) with \( 0 \leq i < j \leq n \) be the least indices such that

\[ \hat{\pi}_i = \hat{\pi}_j. \]

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

Observe:

- \( 1 \leq i,j \leq m \)

- \( 1 \leq j - i \leq n \) (wice \( i < j \))

\[ \delta(q_0, x_{i+1} \ldots x_j) = \hat{\delta}(\hat{\delta}(q_0, x_1), \ldots, x_j) = \hat{\delta}(\hat{\delta}(q_0, x_i), x_{i+1} \ldots x_j) = \hat{\delta}(q_i, x_{i+1} \ldots x_j) = \hat{\delta}(q_m, x_{i+1} \ldots x_j) \]

\[ = \hat{\delta}(q_j, x_{i+1} \ldots x_j) = \cdots = \hat{\delta}(q_j, x_{j-1} x_j) = \hat{\delta}(q_j, x_j) = q_m \in F \Rightarrow x_{i+1} \ldots x_j \in L \]
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 \( m \), and the decomposition \( x, y, z \).

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

for all \( m \)

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

for all decompositions \( w = xy^kz \) of \( w \)

with \( |y| \leq 1 \)

\( |x y| \leq m \)

there exists \( k \geq 0 \) 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 \( m \)
- Ed chooses \( w \in L \) with \( |w| > m \)
- Alice chooses a decomposition \( w = xy^kz \) with \( |y| \geq 1 \)
- Ed chooses \( k \geq 0 \), 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 $L_\text{eq}$ be not regular.

Let's play the game and show that $\text{Ed}$ can always win.

- $\text{Ed}$ chooses $L_\text{eq}$
- Alice chooses some $m$
- $\text{Ed}$ 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 $y \neq \varepsilon$ and $|x \cdot y| \leq m$

  note that, since $|x \cdot y| \leq m$, we have $x \cdot y = 0 \ldots 0$

  $\Rightarrow$ let $x = 0 \ldots 0$

  then $y = 0 \ldots 0$

  $\Rightarrow L_\text{eq}$ is not regular

- $\text{Ed}$ chooses $b = 0$

  then $x \cdot y \cdot z = xz = 0^b 0^m-a-b \cdot 1^m = 0^m 0^m \notin L$

  and $\text{Ed}$ wins.

$\Rightarrow L_\text{eq}$ is not regular.

Exercise: E.4.2

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

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 $\iff$ $\text{Ed}$ has a winning strategy.

Example: Consider $L = L_1 \cdot L_2$ with $L_1$ regular.

We have that $L$ is not regular but $\text{Ed}$ does not have a winning strategy.
Decision problems for regular languages 4/12/2008 4.3

Decision problem: Let \( \mathcal{O} \) be some property of languages

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

Output: does \( L \) have property \( \mathcal{O} \)?

- yes
- no

A decision algorithm decides a decision problem:

- always terminates in finite time

Emptyness: decide if a regular language \( L \) 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 \( \delta(q, \sigma) = p \) for some \( \sigma \), then \( p \) is reachable

For \( n \) states, this takes at most \( O(n^2) \)

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

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

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

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

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

\( \Rightarrow \) 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
  $\implies$ test takes linear time in $|w|$

- when $L$ is given as an NFA $A_N$
  - if we compute the equivalent DFA $\Rightarrow$ exponential in $|A_N|$
    \hspace{1cm} linear in $|w|$

- we can also simulate directly the NFA, by computing the sets of states the NFA is in after each input symbol
  $\Rightarrow O(|w| \cdot \delta^2)$ \hspace{1cm} where $\delta$ is the number of states of $A_N$

  at each step at most $\delta$ states
  each with at most $\delta$ successors

Equality: given regular languages $L_1$, $L_2$
                                   decide whether $L_1 = L_2$

Idea: reduce to emptiness:
consider $L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ \hspace{1cm} (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 $L$
Simplicity: 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:** $\Rightarrow$: Let $w \in L$ with $n \leq |w|$.

By pumping lemma, $w = x \cdot y \cdot z$ with $y \neq \lambda$

and $\forall k > 0, x \cdot y^k \cdot z \in L$.

Since $L$ 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 $< m$).

Let $w$ be the shortest string in $L$ of length $\geq m$.

**Claim:** $|w| < 2m$

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

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

and $x \cdot y^k \cdot z \in L$

We have:

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

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

This contradicts choice of $w$ as shortest string, which gives the claim.

Hence, we have a string $w \in L$ with $n \leq |w| < 2m$

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: Give an algorithm to decide whether a regular
language $L$ is universal, 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 $L_2$.
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:**
\[ p \equiv q \text{ if for all } w \in \Sigma^* : \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \]

The equivalence relation induces a partition of \( Q \)
\[ Q = C_1 \cup C_2 \cup \cdots \cup C_k \]

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

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

\[ p \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \rightarrow \cdots \rightarrow p_{m-1} \xrightarrow{e_m} p_m \]
\[ q \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \rightarrow \cdots \rightarrow q_{m-1} \xrightarrow{e_m} q_m \]

one is final and

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 \( q_j \).

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

(intuitively, there is no inequivalence proof of length \( \leq i \))
The following is immediate to see:

\[ \begin{align*}
\rho \neq i+1 q & \text{ if and only if for some } e \in \Sigma \\
\delta(q,e) & \neq \delta(q',e).
\end{align*} \]

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

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

Step \( i+1 \): determine \( \rho \equiv_{i+1} q \) iff \( \forall e \in \Sigma \)
\[ \delta(q,e) \equiv_i \delta(q',e) \]
compute refined partition.

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

Example:

![Diagram of a state transition graph]

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

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

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

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

Step 4: no change.
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: $\{ C[q_6] \mid q_6 \in F \}$

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

```
Exercise E.4.4
```

**Example:**

```
\begin{tikzpicture}
  \node (C1) at (0,0) {$C_1$};
  \node (C2) at (1,0) {$C_2$};
  \node (C3) at (2,0) {$C_3$};
  \node (C4) at (3,0) {$C_4$};
  \node (C5) at (4,0) {$C_5$};

  \draw[->, thick] (C1) -- node[above] {$1$} (C2);
  \draw[->, thick] (C2) -- node[above] {$0$} (C3);
  \draw[->, thick] (C3) -- node[above] {$1$} (C4);
  \draw[->, thick] (C4) -- node[above] {$0$} (C5);
  \draw[->, thick] (C5) -- node[above] {$1$} (C1);
  \draw[->, thick] (C1) -- node[above] {$0$} (C1);
  \draw[->, thick] (C3) -- node[above] {$0$} (C3);
\end{tikzpicture}
```

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.