Lecture 8 — Properties of Regular Languages (1)

CSE205: Theory of Computation

Minseok Jeon

2026 Fall

Properties of Regular Languages

We have built up a rich theory of regular languages: DFAs, NFAs, $\boldsymbol{\varepsilon}$-NFAs, regular expressions, and regular grammars all describe exactly the same family of languages.

We now study *properties* of this family:

- **Closure properties**: building new regular languages from old ones.
- The **Pumping Lemma** for regular languages (next lecture): a tool for proving that certain languages are *not* regular.

Closure Properties

If one (or several) languages are regular, then certain related languages are also regular. For example:

- Given regular languages $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, the union $\boldsymbol{L}_1 \cup \boldsymbol{L}_2$ is also regular.
- Given regular languages $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, the intersection $\boldsymbol{L}_1 \cap \boldsymbol{L}_2$ is also regular.

We say that *the family of regular languages is closed under union and intersection*.

Closure Properties

The family of regular languages is closed under:

- union
- difference
- complementation
- intersection
- concatenation
- star (Kleene closure)
- reversal
- homomorphism
- ...

We prove each by an explicit *construction* on automata (or regular expressions).

Closure under Union

Theorem

If L and M are regular languages, then so is $L \cup M$.

Proof.

Since L and M are regular, there are regular expressions r and s with $L = L(r)$ and $M = L(s)$. Then $r + s$ is a regular expression and

$$L(r + s) = L(r) \cup L(s) = L \cup M.$$

Hence $L \cup M$ is regular. □

Alternatively: combine the ε-NFAs for L and M with a new start state having ε-transitions to both start states.

Closure under Concatenation and Star

Theorem

If L and M are regular, then so are $L \cdot M$ and L^*.

Proof.

Let r and s be regular expressions with $L = L(r)$ and $M = L(s)$. Then

$$L(rs) = L \cdot M, \qquad L(r^*) = L^*.$$

Since rs and r^* are regular expressions, both languages are regular. □

This is immediate from the equivalence between regular expressions and finite automata.

Closure under Difference

Theorem

If L and M are regular languages, then so is $L - M$.

Proof.

We use the identity

$$L - M = L \cap \overline{M}.$$

Once we show that regular languages are closed under complementation and intersection (next slides), closure under difference follows immediately. □

Closure under Complementation

Theorem

If L is a regular language over alphabet Σ, then $\overline{L} = \Sigma^* - L$ is also regular.

Construction. Let A be a DFA accepting L, i.e. $L = L(A)$ for $A = (Q, \Sigma, \delta, q_0, F)$. Define a DFA B by swapping accepting and non-accepting states:

$$B = (Q, \Sigma, \delta, q_0, Q - F).$$

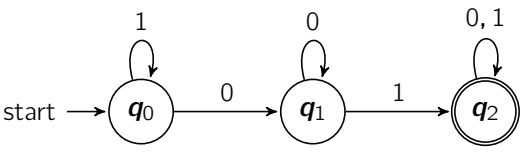
Then B accepts a string w iff A rejects w, so $L(B) = \Sigma^* - L = \overline{L}$.

Note. It is essential that A is a *complete DFA*: every state must have a transition on every symbol (add a dead state if needed). This trick does *not* work directly for NFAs.

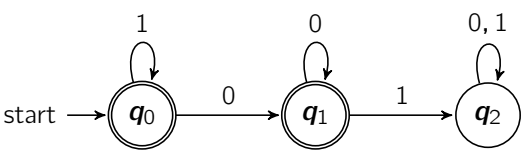
Closure under Complementation: Example

A DFA A for L (left); its complement B swaps final and non-final states (right).

A accepts L



B accepts $\overline{L}$



Closure under Intersection

Theorem

If L and M are regular languages, then so is $L \cap M$.

- **Non-constructive proof.** By De Morgan's law,

$$L \cap M = \overline{\overline{L} \cup \overline{M}}.$$

  Regular languages are closed under complementation and union, so $L \cap M$ is regular.
- **Constructive proof.** Directly construct an automaton (the *product automaton*) that accepts $L \cap M$.

Closure under Intersection: Product Construction

Let $A_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $A_2 = (P, \Sigma, \delta_2, p_0, F_2)$ be DFAs for L and M. Define the *product automaton*

$$A = (Q \times P,\ \Sigma,\ \delta,\ (q_0, p_0),\ F_1 \times F_2)$$

where the two machines are run in parallel:

$$\delta((q, p), a) = (\delta_1(q, a),\ \delta_2(p, a)).$$

A state (q, p) is accepting iff *both* $q \in F_1$ and $p \in F_2$. Therefore

$$L(A) = L(A_1) \cap L(A_2) = L \cap M.$$

For union, instead take final states $\{(q, p) \mid q \in F_1 \text{ or } p \in F_2\}$.

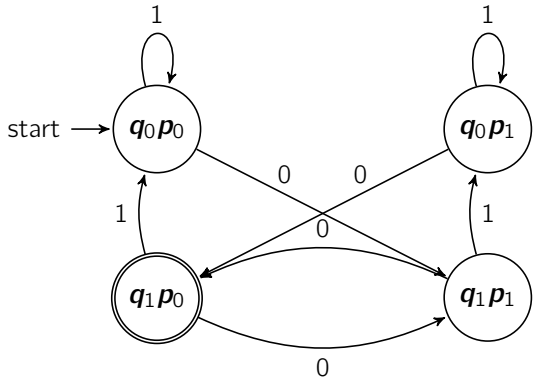
Intersection: Example

A_1 accepts strings ending in 0; A_2 accepts strings with an even number of 0's.

The product automaton has states $(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1)$ and accepts $L(A_1) \cap L(A_2)$.

Intersection: Example — Product Automaton

Writing $q_i p_j$ for the pair state (q_i, p_j), the product automaton for $A_1 \times A_2$ is:



Only $q_1 p_0$ (last symbol 0, even number of 0's) is accepting: the unique pair state where A_1 is in q_1 *and* A_2 is in p_0.

Closure under Reversal

Theorem

If L is a regular language, then so is $L^R = \{ w^R \mid w \in L \}$.

Construction. Let A be an ε-NFA accepting L. Build an automaton for L^R:

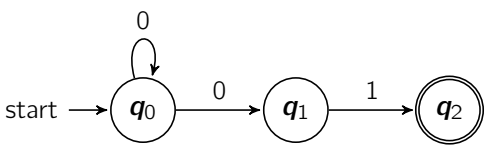
1. Reverse all the arcs in the transition diagram of A.
2. Make the old start state q_0 the *only* accepting state of the new automaton.
3. Create a new start state p_0 with ε-transitions to all the *old* accepting states of A.

A computation of the new machine traces a computation of A backwards, so it accepts w iff A accepts w^R. Hence the new machine accepts L^R.

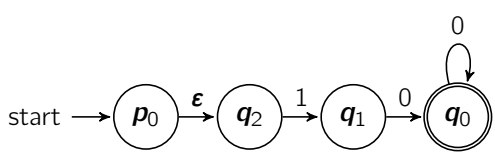
Reversal: Example

A accepts L described by $0^*\,0\,1$ (read 0's then 0 then 1).

A for L



A' for L^R



All arcs are reversed; q_0 becomes the only accepting state; a new start p_0 points by ε to the old accepting state q_2.

Closure under Homomorphism

Definition (Homomorphism)

Suppose Σ and Γ are alphabets. A function

$$h : \Sigma \to \Gamma^*$$

is called a *homomorphism*. It is extended to strings $w = a_1 a_2 \cdots a_n$ by

$$h(w) = h(a_1)\, h(a_2) \cdots h(a_n), \qquad h(\varepsilon) = \varepsilon,$$

and to a language L by $h(L) = \{\, h(w) \mid w \in L \,\}$.

Theorem

If L is a regular language over Σ and h is a homomorphism on Σ, then $h(L)$ is also regular.

Homomorphism: Proof Idea and Example

Proof idea. Take a regular expression $\boldsymbol{r}$ for $\boldsymbol{L}$. Replace every symbol $\boldsymbol{a}$ in $\boldsymbol{r}$ by the string $\boldsymbol{h}(\boldsymbol{a})$; the result is a regular expression for $\boldsymbol{h}(\boldsymbol{L})$.

Example

Let $\Sigma = \{0, 1\}$, $\Gamma = \{\boldsymbol{a}, \boldsymbol{b}\}$, and define

$$\boldsymbol{h}(0) = \boldsymbol{ab}, \qquad \boldsymbol{h}(1) = \boldsymbol{\varepsilon}.$$

This replaces each 0 by $\boldsymbol{ab}$ and erases each 1. For instance,

$$\boldsymbol{h}(0011) = \boldsymbol{ab}\,\boldsymbol{ab}\,\boldsymbol{\varepsilon}\,\boldsymbol{\varepsilon} = \boldsymbol{abab}.$$

If $\boldsymbol{L} = \boldsymbol{L}(1\,0^*\,1)$ (any number of 0's surrounded by 1's), then applying $\boldsymbol{h}$ replaces each 1 by $\boldsymbol{\varepsilon}$ and each 0 by $\boldsymbol{ab}$, giving

$$\boldsymbol{h}(\boldsymbol{L}) = (\boldsymbol{ab})^*.$$

Summary

- The family of regular languages is closed under union, concatenation, star, difference, complementation, intersection, reversal, and homomorphism.
- Most proofs are *constructive*: from machines/expressions for the inputs we build a machine/expression for the result.
  - Complement: swap final / non-final states of a complete DFA.
  - Intersection / union: product (Cartesian) automaton.
  - Reversal: reverse arcs, swap start and accepting roles.
  - Homomorphism: substitute $\boldsymbol{h}(\boldsymbol{a})$ for each symbol in a regular expression.
- **Next:** the Pumping Lemma — a tool to prove that a language is *not* regular.