The class of regular languages is closed under union, intersection, subtraction, complementation, concatenation, Kleene closure and reversal.

Note: closure under union, concatenation and Kleene closure comes from the fact that regular languages are represented by regular expressions; closure under intersection, subtraction and complementation comes from the fact that regular languages are represented by DFAs. The only part still missing is the closure under reversal.

Proof. We could prove it by structural induction (on the number of operators applied), since $\emptyset^R = \emptyset$, $(\{\varepsilon\})^R = \{\varepsilon\}$ and $(a)^R = \{a\}$ for every $a \in \Sigma$, and the extension through union, concatenation and Kleene closure is not hard.

We will prove it by construction, instead:

- Assume $M$ is an NFA (or DFA), with language $L(M)$.
- We construct and NFA $M'$ such that $L(M') = L(M)^R$.
- The equivalence of DFAs and NFAs will do the rest. This is the only proof (of any of the statements of the theorem) where NFAs are really used.

Details: Let $M = (Q, \Sigma, \delta, q_0, F)$. We construct $M'$ as follows: if $|F| > 1$, introduce a new state $s$ and connect every element in $F$ to it via an $\varepsilon$-transition. Otherwise, let $s$ be the single state in $F$.

Define: $M' = (Q \cup \{s\}, \Sigma, \delta', \{s\}, \{q_0\})$, where $q' \in \delta(q, a) \Leftrightarrow q \in \delta'(q', a)$. $\forall a \in \{\varepsilon\} \cup \Sigma$.

All we have done is to "reverse the arrows", after a small adjustment. We have an NFA (we cannot expect a DFA, even if $M$ was a DFA with a singleton accepting state - why?) whose accepting paths are exactly the reversal of the accepting paths of the original.

Closure Properties of Regular Languages - Substitutions

Substitution. Let $f$ be a mapping: $f(a) = L_a$, where $a \in \Sigma$ and $L_a$ is a language over an alphabet $\Gamma$. Extend the function $f$ to $\Sigma^*$ by $f(\varepsilon) = \{\varepsilon\}$ and $f(a_1a_2...a_k) = f(a_1)f(a_2)...f(a_k)$ for $a_1, ..., a_k \in \Sigma$. For any language $L \subseteq \Sigma^*$, we apply $f$ to $L$: $f(L) = \bigcup_{x \in L} f(x)$.

Ex.: $L = \{01, 10\}; f(0) = 0(0+1)^*; f(1) = (0+1)^*$.

$f(L) = \{0(01) \cup f(1) = f(0) f(1) \cup f(1) f(0) \}
= (0+1)^* (0+1)^* (0+1)^* (0+1)^* (0+1)^* (0+1)^* (0+1)^* (0+1)^* (0+1)^*$

A substitution is called a homomorphism if, for any $a \in \Sigma, f(a)$ is a language with a single string (= a singleton string).
Finite Automata

Closure Properties of Regular Languages - Substitutions

Proposition. Let $f$ be a substitution over $\Sigma$; assume that $L \subseteq \Sigma^*$ is a regular language and that $f(a)$ is a regular language for each $a \in \Sigma$. Then $f(L)$ is a regular language.

Proof. Let $r$ be a RE for $L$, $r_a$ a RE for $f(a)$ for each $a \in \Sigma$. Replace each occurrence of $a$ in $r$ by $f(a)$. We obtain a new regular expression $r'$.

We observe:

a) For any two sets $A, B \subseteq \Sigma^*$, $f(A \cup B) = f(A) \cup f(B), f(A \cap B) = f(A) \cap f(B),$ and $f(A^c) = f(A)^c$.

b) For any two regular expressions $r$ and $s$, $(r + s)' = r' + s'$, $(rs)' = r's'$, and $(r^*)' = (r')^*$.

This could be used as part of a detailed proof as set inclusions and RE equalities...

---

Finite Automata

Closure Properties of Regular Languages - Quotients

Definition. The quotient of two languages, denoted by $L_1 / L_2$, is given by $L_1 / L_2 = \{ x \mid \exists y \in L_2 \, \forall z \in L_1 \, (xzy \in L_1) \}$.

Proposition. If $L_i$ is a regular language, then $L_1 / L_2$ is regular.

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting $L_i$. By definition, for any $x \in L_1 / L_2$, $\exists y \in L_2 \, b(q_0, xy) = b(q_0, x) \in F$. If we define $F' = \{(q \in Q) \mid \exists y \in L_2 \, (b(q, y) \in F)\}$, $M = (Q, \Sigma, \delta, q_0, F')$ is a DFA that accepts $L_1 / L_2$.

Note: The proof is not quite constructive, at the moment, since we would need to show that a finite number of strings in $L_2$ suffices to construct $F'$.
Finite Automata

Closure Properties of Regular Languages - Examples

Ex. 2.37. Let \( L \) be regular over \( \Sigma, k \) a pos. integer and \( \phi : \Sigma^k \to \Sigma \). Then \( L_1 = (\phi(a_1, a_2, \ldots, a_k) \cdots \phi(a_{k-1}, a_k) a \in L) \) is regular. Note that a string of length \( nk \) in \( L \) becomes a string of length \( n \) in \( L_1 \).

Proof. Let \( L = L(M) \) for \( M = (Q, \delta, s, F) \). Let \( M' = (Q, \delta', s, F) \), with \( \forall (q \in Q, a \in \Sigma) \delta'(q, a) = (\delta(q, a_1, \ldots, a_k) a = a) \).

Note: there is no requirement for \( \phi \) to be 1-1.

We can see that \( L_1 = L(M') \).

Note that \( \delta(q, a_1, \ldots, a_k) \) is the state of \( M \) reached from \( q \) after consuming the string \( a_1 \cdots a_k \).

Ex. 2.38. If \( L \) is regular, then \( \text{MIN}(L) \) is regular.

Proof. Let \( L = L(M) \), where \( M = (Q, \delta, s, F) \) is a DFA. Let \( M' \) be the NFA obtained from \( M \) by deleting all the out-edges from the final states.

It is clear that \( M' \) accepts \( \text{MIN}(L) \).

Ex. 2.39. Let \( A \) and \( B \) be regular over \( \{0, 1\} \). Then \( A \lor B = \{x \lor y \mid x \in A, y \in B, |x| = |y| \} \) is regular.

Proof. Let \( M_A = (Q_A, \delta_A, s_A, F_A) \) and \( M_B = (Q_B, \delta_B, s_B, F_B) \) be the respective DFAs. We build a product NFA \( M' \) as follows:

\[
M' = (Q_A \times Q_B, \{0, 1\}, \delta', s', F) \text{ where } \\
\delta'(q_A, q_B) = (\delta_A(q_A, 0), \delta_B(q_B, 0)) \\
\delta'(q_A, q_B) = (\delta_A(q_A, 1), \delta_B(q_B, 1)), \delta_A(q_A, 0), \delta_B(q_B, 0)), \delta_A(q_A, 1), \delta_B(q_B, 1)).
\]

Minimum Deterministic Finite Automata

We have seen that regular languages give rise to at least three different, and equivalent, notational devices that represent them. For one of them, at least, the following question is meaningful: does a regular language have a corresponding (unique) minimal DFA, in the sense that no other DFA accepting the language has fewer states? If the answer is yes, can we construct it?

The constructions we have, from REs to NFAs to DFAs tend to "blow up" the number of states, up to exponential cardinality in the number of states of the NFA - which already "bloats" the number of atomic terms in the regular expression (digraph construction).

We could use a different characterization of regular languages.
Finite Automata

Minimum Deterministic Finite Automata - the Index

Definition. For any language \( L \subseteq \Sigma^* \), we define a relation \( R_L \) on \( \Sigma^* \):
\[
x R_L y \iff (\forall w \in \Sigma^*) [xw \in L \iff yw \in L].
\]

Proposition. \( R_L \) is an equivalence relation on \( \Sigma^* \):
1) It is reflexive: (\( \forall x \in \Sigma^* \)) \( x R_L x \)
2) It is symmetric: (\( \forall x, y \in \Sigma^* \)) \( x R_L y \Rightarrow y R_L x \)
3) It is transitive: (\( \forall x, y, z \in \Sigma^* \)) \( x R_L y, y R_L z \Rightarrow x R_L z \)

Proof. An easy exercise.

Corollary. \( R_L \) partitions \( \Sigma^* \) into disjoint equivalence classes, where the class containing \( x \) is denoted by \( [x]_{R_L} \).

Proof. Another easy exercise.

Definition. The number of equivalence classes of \( R \) in \( \Sigma^* \) is called the index of \( R \) and denoted by \( \text{Index}(R) \).

Minimum Deterministic Finite Automata - the Index - Example

Ex. 2.46: Let \( L \) be the set of binary strings starting and ending with the same symbol. Find all equivalence classes of \( R_L \) (in \( \Sigma^* \)).

Soln.: a) we first show that \( \forall x, y \in \Sigma^*, x R_L y \iff x \) and \( y \) start with the same symbol and end with the same symbol. ⇒ By contradiction. Assume \( x \) and \( y \) start with different symbols, with, say, \( x \) starting with a 0 and \( y \) starting with a 1. Then \( x0 \in L \) while \( y0 \notin L \), so that \( x R_L y \) does not hold. This implies that \( x \) and \( y \) must start with the same symbol. Note further that \( x R_L y \Rightarrow x \in L \iff y \in L \) (\( \epsilon \in \Sigma^* \)), and therefore, since they start with the same symbol, they must end with the same symbol.

We can now characterize the (5) equivalence classes:
\[
\begin{align*}
[x]_{R_L} &= \epsilon, \\
[0]_{R_L} &= 0 + 0(0 + 1)^*0, \\
[01]_{R_L} &= 0(0 + 1)^*1, \\
[1]_{R_L} &= 1 + 1(0 + 1)^*1, \\
[10]_{R_L} &= 1(0 + 1)^*0
\end{align*}
\]
conclude that:

An argument similar to those already encountered allows us to

1) We still have to place two sets of strings: the ones denoted by

\[ 0(0+1)^*111, 0(0+1)^*0, 1(0+1)^*0 \]

\[ \varepsilon, \text{ otherwise} \]

2) \[ \varepsilon \] is any other nontrivial string, it must be one of

\[ 0(0+1)^*0, 0(0+1)^*1, 1(0+1)^*0, 1(0+1)^*1 \]

Then \( x \in L \) or \( 0(0+1)^*011, 0(0+1)^*111 \notin L \). Similarly for \( \varepsilon 00 \) and \( 1(0+1)^*000, 1(0+1)^*000 \).

So \( [\varepsilon]_R \) contains just one string: \( \varepsilon \).
**Finite Automata**

### Minimum DFAs - the Index - Second Example

**Problem 2.7.1.a:** find all equivalence classes of \( R_s \) for the language 
\[ L = (0 + 1)^*0(0 + 1)^*. \]

Claim: \([0]_{R_s} = \ldots = [0^n]_{R_s} = \ldots = [1^*0^n]_{R_s} \]

for \( m \geq 1, n \geq 1 \), i.e., \( w \) must contain a 1 for the concatenation to be in \( L \).

Claim: \([1]_{R_s} = [0]_{R_s} \) from the previous slide.

Claim: \([0]_{R_s} = [1]_{R_s} \).

All other strings in \( \Sigma^* \) belong to one of the classes just constructed.

*Automaton for \( L \):*

![Automaton Diagram]

---

### Finite Automata

**Minimum DFAs - the Index - Third Example**

**Problem 2.7.2.a:** show that for \( L = \{0^n1^* \mid 0 \leq m \leq n \} \), \( \text{Index}(L) = \infty \).

**Soln:** first observe that \( \Sigma = \{0, 1\} \), \( R_s \) is defined over \( \Sigma^* \) (= all strings over \( \{0, 1\} \)) and, specifically,

\[ x \circ y \Leftrightarrow (w \in \Sigma^*) \{xw \in L \Leftrightarrow yw \in L\} \]

for any two strings \( x, y \in \Sigma^* \).

Start with the class of the empty string, \([\epsilon]_{R_s} \); for what \( y \in \Sigma^* \) is it true that \( xw \in L \Leftrightarrow yw \in L \)? It is clear that \( xw \in L \Leftrightarrow w \in L \), so that \( w = 0^*1^n \) for some \( 0 \leq m \leq n \).

**Question:** what must \( y \) look like so that \( yw \in L \) for all \( w \) of the form \( 0^*1^n \), \( 0 \leq m \leq n \), and only for those \( w \)? Assume \( y = \epsilon^k \) for some \( k \geq 0 \). Then \( 0^*0^m \epsilon^k \in L \iff 0^m w \in L \) \( (0 \leq m \leq n < k + m) \). Similar arguments hold for any non-empty strings: \([\epsilon]_{R_s} = \{\epsilon\} \).

---

**Finite Automata**

**Minimum Deterministic Finite Automata**

**Lemma.** Let \( L \) be accepted by the DFA \( M = (Q, \Sigma, \delta, q_0, F) \), and let \( \delta \) be the extended transition function of \( M \). Then, for any strings \( x, y \in \Sigma^* \),

\[ \delta(x, x) = \delta(x, y) \Rightarrow x \circ y \]

**Proof.** If \( \delta(x, x) = \delta(x, y) \), then, for any \( w \in \Sigma^* \),

\[ \delta(x, xw) = \delta(x, x), w) = \delta(x, y, w) = \delta(x, yw) \]

Thus, for any \( w \in \Sigma^* \), \( xw \in L \Leftrightarrow yw \in L \).

**Note.** The lemma shows that, if \( L \) is accepted by a DFA \( M \) of \( n \) states, all strings with the same ending state are in the same equivalence class of \( R_s \). In particular \( \text{Index}(R_s) \leq n \), and is thus finite. If we can find a DFA accepting \( L \) and with exactly \( \text{Index}(R_s) \) states, we have found the desired minimum DFA.
Finite Automata

Minimum Deterministic Finite Automata

Theorem 2.48 (Myhill-Nerode). For any regular language \( L \), its minimum DFA has exactly \( \text{Index}(R_j) \) states.

Proof. Let \( L \) have alphabet \( \Sigma \). Define \( M = (Q, \Sigma, \delta, s, F) \) by:

1. \( Q = \{ [x]_M \mid x \in \Sigma^* \} \) (and \( |Q| < \infty \) if \( L \) is regular);
2. \( \delta([x]_M, a) = [xa]_M \), for any \( a \in \Sigma \);
3. \( s = [\varepsilon]_M \);
4. \( F = \{ [x]_M \mid x \in L \} \).

The function \( \delta \) is well-defined since:

\[
[x]_M = [y]_M \quad \Rightarrow \quad [xa]_M = [ya]_M \quad \Rightarrow \quad \delta([x]_M, a) = [xa]_M = \delta([y]_M, a).
\]

By induction on \(|y| \) we can extend this to \( \delta([x]_M, y) = [x]_M \quad \forall y \in \Sigma^* \), proving that \( L(M) = \{ x \in L \mid [x]_M \in F \} \Leftrightarrow \delta([x]_M, y) \in F \Rightarrow M \) accepts \( x \). Since \( M \) has \( \text{Index}(R_j) \) states, it is, by the previous Lemma, a min. DFA for \( L \).

Finite Automata

Minimum Deterministic Finite Automata

Corollary. A language \( L \) is regular \( \iff \text{Index}(R_j) < \infty \).

Proof. No use of regularity (other than for the finiteness of \( Q \)) was made in the Myhill-Nerode theorem. We can thus construct the minimum DFA for any language as long as \( \text{Index}(R_j) < \infty \). Such languages \( L \) must be regular.

Note: the finiteness of DFAs allows us to obtain a "simplification" - we don't need to look at all of \( \Sigma^* \) to characterize \( R_j \), but only to strings up to a certain length.

Finite Automata

Minimum Deterministic Finite Automata

Proposition. \( L \) is regular \( \iff \exists k \in \mathbb{N} \ s.t. \{ x \in \Sigma^* \mid |x| \leq k, z_c \in L \Leftrightarrow z \in L \} \).

Proof. \( \Rightarrow \) Let \( M = (Q, \Sigma, \delta, s, F) \) be a DFA accepting \( L \); let \( k = |Q| - 1 \).

\( \Rightarrow \) This is obvious, since \( s \in R_j \Leftrightarrow \forall z \in F \Leftrightarrow \forall z \in L \Leftrightarrow |z| \leq k \).

\( \Leftarrow \) Consider the product DFA \( M' = M \times M \). Let \( = \in \Sigma^* \) with \( |z| > k \). Let \( x, y \in \Sigma^* \). The computation path of \( M' \) on \( \Rightarrow \) starting from \( q'_{x} = (b(x, x), b(x, y)) \), to \( q''_{y} = (b(x, xw), b(x, yw)) \), contains at least \( |Q^2| + 1 \) states, while \( M' \) has \(|Q^2| \) states, and therefore it contains some cycles in the transition diagram of \( M' \). Eliminate the cycles and keep only a single path from \( q'_{x} \) to \( q''_{y} \), which corresponds to the computation path for a string \( z \) with \( |z| \leq k \), so that \( q_{x} = (b(x, x), b(x, y)) \). By assumption, \( x \in L \Leftrightarrow y \in L \]: the states \( b(x, x) \) and \( b(x, y) \) are both in \( F \) or not in \( F \).

So \( xw \in L \Leftrightarrow yw \in L \), and the result follows.
The classes are to do two things

Note: we have developed some results and techniques which allow us to do two things
a) we have a way of constructing a minimum DFA for a regular language;
b) we have a way of determining (maybe) whether a language is regular or not (being unable to construct a DFA is quite different from proving that no DFA exists) by computing Index($R_i$).

Minimum Deterministic Finite Automata - Example

Ex. 2.51.: find the minimum DFA for the language $L = (0 + 1)^*01$.

Sols.: we compute the equivalence classes of $R_i$ (some “bit analysis”), beginning from the shortest string in $\Sigma$.

1. $[1]_{R_i} = \{x \in (0, 1)^*(\forall w \in (0, 1)^*(\forall w \in (0 + 1)^*01 \iff w \in (0 + 1)^*01)) \}$ We need to characterize $x \in [1]_{R_i}$. What are the possible relevant endings of $x$? 0, 1, 00, 01, 10, 11. There is no point in looking at longer strings, since $i$ fixes the last two characters. $x$ may not end with 0 (or 01); if it did $x$ (or $xx$) would be in $(0 + 1)^*01$, which would imply that $w = 1$ (or $w = x$) is in $(0 + 1)^*01$. Conversely, if $x$ does not end with 0 or 01, then $x \in [1]_{R_i}$. Thus $[1]_{R_i} = \{x \in (0, 1)^* | \forall w \in (0, 1)^*(\forall w \in (0 + 1)^*01 \iff w \in (0 + 1)^*01)) \}$.

2. $[0]_{R_i} = \{x \in (0, 1)^* | (\forall w \in (0, 1)^*(0w \in (0 + 1)^*01 \iff xw \in (0 + 1)^*01)) \}$. What must $x$ end with? Since, for $w = 1$, $0w \in (0 + 1)^*01$), $x$ must end with 0. Conversely, if $x$ ends with 0, $x \in [0]_{R_i}$. Thus $[0]_{R_i} = (0 + 1)^*0$.

The two classes must coincide (by $R_i$ being an equivalence relation).

Minimum Deterministic Finite Automata

Ex. 2.53. Find the minimum DFA equivalent to the DFA:

![Minimum Deterministic Finite Automata - Example](image-url)
Finite Automata

Minimum Deterministic Finite Automata - Example

Soln. 1: \( M = (Q, \Sigma, \delta, q_0, F) \), \( Q = \{ q_0, q_1, q_2, q_3, q_4, q_5 \} \), \( L = L(M) \).

Recall: for states \( p \) and \( q \),
\( p R_q L q \Leftrightarrow S_p = \{ x \in \Sigma^* \mid \delta(s, x) = p \} \),
\( S_q = \{ x \in \Sigma^* \mid \delta(s, x) = q \} \).

To find \( R_q \), between any two states, construct a graph \( G \):
Each vertex is an unordered pair \( (q_i, q_j) \).
Let \( U \) be the set of vertices \( (q_i, q_j) \) with one vertex \( q_i \in F \) and the other vertex \( q_j \not\in F \). For each vertex \((q_i, q_j) \not\in U \) (either both vertices are in \( F \) or not in \( F \)), with \( i \neq j \), draw edges \((q_i, q_j) \Leftrightarrow (\delta(q_i, a), \delta(q_j, a)) \forall a \in \Sigma \).

Claim: \( q_i R_q q_j \Leftrightarrow \) there is no path in \( G \) from \((q_i, q_j) \) to a vertex in \( U \).

How do we use this construction? Construct \( G \).
\( U \) consists of the pairs \((0, 2), (1, 3), (1, 2), (0, 3), (2, 4), (3, 4), (4, 5) \).
The complement of \( U \) consists of \((0, 0), (0, 1), (1, 1), (0, 4), (1, 4), (2, 2), (2, 3), (3, 3), (2, 5), (3, 5), (4, 4), (5, 5) \).

Note: there is no point in starting from a \((q_i, q_i) \) node - why?
From \((0, 1) \):

\begin{align*}
(0, 1) & \quad 2, 3 \quad 1 \\
2 & \quad 3 \quad 1, 4 \\
5 & \quad 4, 5 \quad 0, 1
\end{align*}

The next node in the list is \((0, 4) \):

Note that from the nodes \((0, 4), (1, 4), (2, 5) \) and \((3, 5) \) we can reach a node in \( U \). When we put together the two partial graphs (constructed this way for convenience, we get:

The minimal automaton is exactly the one corresponding to the graph with only black nodes.
Finite Automata

Minimum Deterministic Finite Automata - Example

We finish with just the minimal DFA, where we have highlighted it final nodes:

0
(0, 1)
(1, 3)
1
(4, 4)
(5, 5)
0
1
0
1
0
1
0
1
0
1
0
1

Soln. 2: The critical step of soln. 1 is to determine, for a pair \((q_i, q_j)\) whether there is a path in \(G\) from it to a vertex of \(U\). You need only study pairs where \(i \neq j\), since you know the \((q_i, q_i)\) belong to the same class. Create a table of pairs, marking all pairs in \(U\) with a 0:

Of the unmarked pairs, mark with a 1 those for which there is an \(a \in \Sigma\) taking them to a pair marked with a 0.

Keep on iterating until no more markings. The pairs left unmarked must satisfy \(q_i \text{ R}^* \text{ L} q_j\). Add the \((q_i, q_j)\) needed to complete the graph.

Soln. 3: Split states into \(F\) and \(Q \setminus F\). Chase states from \(F\): if any exit \(F\), break \(F\) into an appropriate number of sub-blocks; same for \(Q \setminus F\). Keep on iterating until no new blocks appear.

Of the unmarked pairs, mark with a 1 those for which there is an \(a \in \Sigma\) taking them to a pair marked with a 0.