REGULAR LANGUAGES

Frank Tsai

December 5, 2023

Contents

Contents **1**

1 Deterministic finite automaton 1

2 Regular expressions 3

1. DETERMINISTIC FINITE AUTOMATON

Definition 1.1. A *deterministic finite automaton* (DFA) consists of the following data

1. a nonempty finite set Q of *states*;
2. a nonempty finite set Σ of *alphabet*;
3. a *transition function* $\delta : Q \times \Sigma \to Q$;
4. an *initial state* $q_0 \in Q$;
5. a set $F \subseteq Q$ of *final states*.

(1·1) A DFA is a simple *abstract machine*. The transition function $\delta : Q \times \Sigma \to Q$ specifies how the machine steps given an input and the current state.

(1·2) Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$. We define a new function $\delta^* : \Sigma^* \to Q$ by recursion:

1. If the input is the empty string then $\delta^*(\varepsilon) = q_0$;
2. If the input is a string s' followed by a character c then $\delta^*(s' \cdot c) = \delta(\delta^*(s'), c)$.

For any string s over the alphabet Σ, $\delta^*(s)$ is the state of D after processing s.

Definition 1.2. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that a string s over the alphabet Σ is *accepted* by D if

$$\delta^*(s) \in F$$

In other words, a string s is accepted by D if D ends up in a final state after processing s.

Definition 1.3. Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The *language* of D, denoted $\mathscr{L}(D)$ is the set

$$\mathscr{L}(D) = \{\, s \in \Sigma^* \mid \delta^*(s) \in F \,\}$$

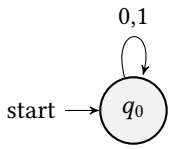
We say that a language $L \subseteq \Sigma^*$ is *recognized* by a DFA D if $\mathscr{L}(D) = L$.

Definition 1.4. A language L is said to be *regular* if there is a DFA D that recognizes L.

(1·3) In the following examples, take $\Sigma = \{\, 0, 1 \,\}$.

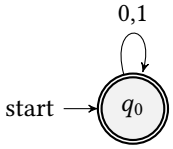
Example 1.5. The empty language $\varnothing$ is regular.

(1·4) This language is recognized by the following DFA:



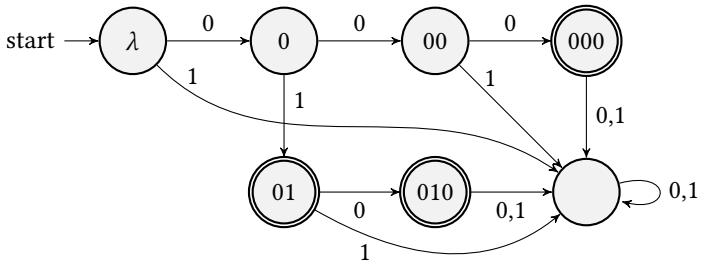
Example 1.6. The language Σ^* is regular.

(1·5) This language is recognized by the following DFA:



Example 1.7. Every finite language L_{Fin} is regular.

(1·6) The idea is that we can construct a DFA by brute force. This works because there are only finitely many strings in L_{Fin}, i.e., finitely many states suffice. For instance, suppose that $L_{\mathsf{Fin}} = \{\, 000, 010, 01 \,\}$. The following DFA recognizes L_{Fin}:



Definition 1.8. Let $L, L' \subseteq \Sigma^*$ be languages. The concatenation $L \cdot L'$ is the language

$$L \cdot L' = \{\, s \cdot s' \in \Sigma^* \mid s \in L \wedge s' \in L' \,\}$$

Theorem 1.9. *The class of regular languages is closed under intersection, union, complement, concatenation, and Kleene closure. That is, if L and L' are regular languages then so are $L \cap L'$, $L \cup L'$, $\Sigma^* \setminus L$, $L \cdot L'$, and L^*.*

Proof sketch. **(1·7)** We only sketch the proof of closure under union. Let $D = (Q, \Sigma, \delta, q_0, F)$ (resp., $D' = (Q', \Sigma, \delta', q_0', F')$) be a DFA that recognizes L (resp., L'). The main idea is to construct a new DFA that simulates both D and D':

1. take $Q \times Q'$ as the set of states;
2. take Σ as the alphabet;
3. define the transition function $\Delta : (Q \times Q') \times \Sigma \to Q \times Q'$ by
$$\Delta((q, q'), c) = (\delta(q, c), \delta'(q', c))$$
4. take $(q_0, q_0') \in Q \times Q'$ as the initial state;
5. take $\{ (q, q') \in Q \times Q' \mid q \in F \vee q' \in F' \}$ as the set of final states.

□

2. REGULAR EXPRESSIONS

Definition 2.1. Given an alphabet Σ, *regular expressions* are defined inductively as follows:

1. $\varnothing$ is a regular expression;
2. ε is a regular expression;
3. each literal $c \in \Sigma$ is a regular expression;
4. if r and r' are regular expressions then $r \cdot r'$ is a regular expression;
5. if r and r' are regular expressions then $r + r'$ is a regular expression;
6. if r is a regular expression then r^* is a regular expression.

(2·1) We have just defined the syntax of regular expressions. The semantics of regular expressions is defined by mapping each regular expression to a language.

Definition 2.2. The language of a regular expression r is defined by recursion on r:

1. the language of the regular expression $\varnothing$ is the empty language:
$$\mathscr{L}(\varnothing) = \varnothing$$
2. the language of the regular expression ε is the language containing just the empty string:
$$\mathscr{L}(\varepsilon) = \{ \varepsilon \}$$
3. the language of the regular expression c is the language containing just c: for each $c \in \Sigma$,
$$\mathscr{L}(c) = \{ c \}$$
4. the language of the regular expression rr' is the concatenation of $\mathscr{L}(r)$ and $\mathscr{L}(r')$:
$$\mathscr{L}(r \cdot r') = \mathscr{L}(r) \cdot \mathscr{L}(r')$$

5. the language of the regular expression $r + r'$ is the union of $\mathscr{L}(r)$ and $\mathscr{L}(r')$:

$$\mathscr{L}(r + r') = \mathscr{L}(r) \cup \mathscr{L}(r')$$

6. the language of the regular expression r^* is the Kleene closure of $\mathscr{L}(r)$:

$$\mathscr{L}(r^*) = (\mathscr{L}(r))^*$$

Theorem 2.3. *Let e be a regular expression, $\mathscr{L}(e)$ is regular, i.e., there is a DFA D that recognizes $\mathscr{L}(e)$.*

Proof. **(2·2)** We proceed by induction on the structure of e. When e is $\emptyset, \varepsilon$, or c, $\mathscr{L}(e)$ is finite. In Example 1.7, we informally argued that finite languages are regular.

(2·3) When e is $r \cdot r'$ or $r + r'$, the induction hypothesis states that $\mathscr{L}(r)$ and $\mathscr{L}(r')$ are regular. By definition $\mathscr{L}(r \cdot r') = \mathscr{L}(r) \cdot \mathscr{L}(r')$ and $\mathscr{L}(r + r') = \mathscr{L}(r) \cup \mathscr{L}(r')$. By Theorem 1.9, the class of regular languages is closed under concatenation and union, so $\mathscr{L}(r \cdot r')$ and $\mathscr{L}(r + r')$ are regular.

(2·4) Finally, when e is r^*, the induction hypothesis again states that $\mathscr{L}(r)$ is regular. By definition, $\mathscr{L}(r^*) = (\mathscr{L}(r))^*$. By Theorem 1.9, the class of regular languages is closed under Kleene closure, so $\mathscr{L}(r^*)$ is regular.

□

(2·5) Theorem 2.3 shows that every regular expression has a corresponding DFA. The converse is also true: every DFA has a corresponding regular expression. The proof of this fact is not trivial.