Regular Languages

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Regular Languages

- The regular languages have been defined to be those languages accepted by a dfa (or nfa, since they are equivalent).
- It could be time-consuming and tricky trying to build a dfa for a language. Therefore it would be nice to have some terse characterizations.
- Such characterizations do exit. We introduce the **regular expressions** and the **regular grammars**.

Regular Expressions

- A regular expression represents a set of strings. It does so by using an alphabet, parentheses, and the operators $+$, $\cdot$, and $*$.
  - $\{a\}$ is denoted by a.
  - $\{a, b\}$ is denoted by $a + b$ (union).
  - $\{ab\}$ is denoted by $a \cdot b$ (concatenation).
  - $\{\lambda, a, aa, aaa, \dots\}$ is denoted by a^* (star-closure).
  - Parentheses are used for grouping purposes.
- For example,

$$(a + (b \cdot c))^* = \text{the star-closure of } \{a\} \cup \{bc\}$$
$$= \{\lambda, a, bc, aa, abc, bca, bcbc, \dots\}.$$

Formal Definition

- Let Σ be an alphabet. Then
  - $\emptyset, \lambda$ and $a \in \Sigma$ are regular expressions. They are called **primitive** regular expressions.
  - If r_1 and r_2 are regular expressions, then so are $r_1 + r_2,\ r_1 \cdot r_2,\ r_1^*$ and (r_1).
  - r is a regular expression if and only if it can be derived from primitive regular expressions by a finite number of applications of $+$, $\cdot$, $*$, and parenthesis.

Language of Regular Expression

- The language of a regular expression r is defined by
  - basic rules

$$\begin{cases} L(\emptyset) = \emptyset \\ L(\lambda) = \{\lambda\} \\ L(a) = \{a\} \end{cases}$$

  - recursive rules

$$\begin{cases} L(r_1 + r_2) = L(r_1) \cup L(r_2) \\ L(r_1 \cdot r_2) = L(r_1)L(r_2) \\ L(r^*) = (L(r))^* \\ L((r)) = L(r) \end{cases}$$

$L(r)$ Is Regular

- If r is a regular expression, then $L(r)$ is a regular language.
- That is, we can construct a dfa or nfa to accept $L(r)$.
- To prove, let r be a regular expression.
  - If r is primitive, then $L(r)$ is regular since $\emptyset, \lambda, \{a\}$ are all regular. The automata are shown in Figure 3.1.
  - If r is not primitive, then r must be derived by a finite number of applications of $\cdot,\ +$ and *. For each of these operations, we can construct an nfa to accept the new language from the nfas for the old languages. Therefore $L(r)$ is regular.

A Regular L is $L(r)$

- Is the converse true? That is, is it true that for every regular language there exists a regular expression?
- More precisely, can we find a regular expression to generate the labels of all walks from q_0 to any final state, given a fa?
- We will show this to be true by using the **generalized transition graphs** (GTG).

Generalized Transition Graphs

- A GTG is a transition graph whose edges are labeled by regular expressions, rather than symbols.
- The label of a walk from the initial state to a final state is a concatenation of regular expressions so is itself a regular expression.
- The language of a GTG is the union of languages represented by the labels (regular expressions) of such walks. It can be represented by a regular expression.
- Every regular language has an nfa. Every nfa has a transition graph which is a special case of GTG. Every GTG has a regular expression.
- Figure 3.8 is a simple example.

Complete GTG

- Two GTGs are equivalent if they accept the same language.
- Given a GTG G, we can create a sequence of increasingly simple GTGs equivalent to G. Eventually, we end up with an equivalent GTG with two states.
- It will be convenient to introduce **complete** GTG, which is a graph with all edges present. Thus, a complete GTG with $|V|$ vertices has $|V|^2$ edges.
- Figure 3.9 illustrate how to make a GTG complete.

Two-State GTG

- For a complete two-state GTG, say Figure 3.10, the regular expression that covers all possible walks is

$$r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

- A three-state GTG has an equivalent two-state GTG as illustrated in Example 3.10. Essentially, we modify the labels of edges not incident on q_2 and then remove all other edges.
- For a GTG with more than 2 states, we can remove one state at a time using the following procedure.

Procedure nfa-to-rex

1. Start with an nfa M with a single final state.
2. Convert M to a complete GTG G. The edge from q_i to q_j is labeled by r_{ij}.
3. If there are only two states, q_i initial and q_j final, G has the regular expression $r_{ii}^* r_{ij}(r_{jj} + r_{ji} r_{ii}^* r_{ij})^*$.
4. If there are three states, q_i initial and q_j final and q_k, remove q_k and associated edges after modifying labels of edges between $\alpha, \beta = i, j$ to be $r_{\alpha\beta} + r_{\alpha k} r_{kk}^* r_{k\beta}$.
5. For four or more states, pick a state q_k to remove. Apply the above rule for all state pairs $q_\alpha, q_\beta \neq q_k$.
6. Repeat 3-5 if $|V| > 2$.

Right-linear Grammars

- A grammar is said to be **linear** if there is only one variable, say B, on the right side.
- A grammar $G = (V, T, S, P)$ is said to be **right-linear** if all productions are of the form

$$A \to xB,$$
$$A \to x,$$

  where $A, B \in V$ and $x \in T^*$.
- Notice the variable on the right side appears in the end.

Language of a Regular Grammar

- If $G = (V, T, S, P)$ is right-linear, then $L(G)$ is regular.
- We can construct an nfa M for $L(G)$ as follows.
  - Let $V_0 = S$. For variable V_i, we construct a state q_i. In addition, we construct a final state q_f.
  - We construct δ such that

$$
\begin{aligned}
V_i \to vV_j \quad &\Rightarrow \quad \delta^*(q_i, v) = q_j \\
V_i \to u \quad &\Rightarrow \quad \delta^*(q_i, u) = q_f
\end{aligned}
$$

- If $S \overset{*}{\Rightarrow} w$, then there exists a walk labeled w from q_0 to q_f by the construction. So $w \in L(G) \Rightarrow w \in L(M)$. If there exists a walk from q_0 to q_f labeled w, then by following q_i in the walk, we have a derivation $S \overset{*}{\Rightarrow} w$.

Grammar of a Regular Language

- If L is regular, then there exists a right-linear grammar G such that $L = L(G)$.
- Let $M = (Q = \{q_0, \ldots, q_n\}, \Sigma = \{a_1, \ldots, a_m\}, \delta, q_0, F)$ be a dfa that $L(M) = L$. We can construct $G = (V, \Sigma, S, P)$ for L as follows.
  - Associate variable V_i for q_i, for all i.
  - If $\delta(q_i, a) = q_j$, then P has a rule $V_i \to aV_j$.
  - If $q_k \in F$, then P has a rule $V_k \to \lambda$.
- If there exists a walk from q_0 to $q_k \in F$ labeled w, then $S \stackrel{*}{\Rightarrow} w$ by the construction. So $w \in L(M) \Rightarrow w \in L(G)$. If $S \stackrel{*}{\Rightarrow} w$, then by this derivation we can find a walk from q_0 to $q_k \in F$. So $w \in L(G) \Rightarrow w \in L(M)$.

Left-linear Grammars

- A left-linear grammar is similarly defined, i.e.

$$A \to Bx, \; A \to x, \text{ where } A, B \in V \text{ and } x \in T^*.$$

- If G is left-linear, by reversing the right side of every production rule, we can construct a right-linear $\widehat{G}$ such that

$$L(\widehat{G}) = (L(G))^R.$$

  L^R is the reversal of L.

- Since regular languages are closed under reversal, we conclude that a left-linear grammar also generates a regular language.

Basic Questions

- How general is the set of regular languages?
  - Given a language (a set of strings), is it always possible to build a dfa for it?
  - If the answer is no, can we identify those languages that are not regular?
- A language is a set. What can we say about the language created by basic set operations, such as union and intersection, on regular languages?
- In answering these questions, we will introduce some properties of regular languages.

Closure Properties

- By **regular set**, we mean the set of languages that are regular. Any regular language is in the regular set.
- The regular set is **closed** under the following operations.
  - union, intersection, concatenation, complementation, and star-closure
  - reversal
  - homomorphism
  - right quotient

Union

- Let L_1, L_2 be regular languages, then $L_1 \cup L_2$ is regular.
- To prove, let $L_1 = L(r_1)$ and $L_2 = L(r_2)$. Since $r_1 + r_2$ is a regular expression, $L(r_1 + r_2)$ is a regular language. By definition,

$$L(r_1 + r_2) = L(r_1) \cup L(r_2) = L_1 \cup L_2.$$

  So $L_1 \cup L_2$ is regular.
- Alternatively, one can construct an automaton for $L_1 \cup L_2$. So the language is regular.

Intersection

- Let L_1, L_2 be regular languages, then $L_1 \cap L_2$ is regular.
- Let $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $M_2 = (P, \Sigma, \delta_2, p_0, F_2)$ be the dfa's for L_1 and L_2. Let $\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\delta}, (q_0, p_0), \widehat{F})$, with

$$\widehat{Q} = Q \times P, \quad \widehat{F} = \{(q, p) : q \in F_1, p \in F_2\},$$

$$\widehat{\delta}((q_i, p_j), a) = (q_k, p_l) \Leftrightarrow \delta_1(q_i, a) = q_k \;\wedge\; \delta_2(p_j, a) = p_l.$$

  Then $L_1 \cap L_2 = L(\widehat{M})$.
- For an indirect proof, note that

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

Concatenation

- Let L_1, L_2 be regular languages, then $L_1 L_2$ is regular.
- To prove, let $L_1 = L(r_1)$ and $L_2 = L(r_2)$. Since $r_1 \cdot r_2$ is a regular expression, $L(r_1 \cdot r_2)$ is a regular language. By definition,

$$L(r_1 \cdot r_2) = L(r_1)L(r_2) = L_1 L_2.$$

So $L_1 L_2$ is regular.

Complementation

- Let L be a regular language, then $\overline{L} = U - L$ is regular.
- To prove, suppose $M = (Q, \Sigma, \delta, q_0, F)$ is the dfa that accepts L. Let $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$. Then any string accepted by M is not accepted by $\overline{M}$, and any string not accepted by M is accepted by $\overline{M}$. So

$$L(\overline{M}) = \overline{L}.$$

Star-Closure

- Let L be a regular language, then L^* is regular.
- To prove, let $L = L(r)$. Since r^* is a regular expression, $L(r^*)$ is a regular language. By definition,

$$L(r^*) = (L(r))^* = L^*.$$

So L^* is regular.

Reversal

- If L is regular, then so is L^R.
- First, note that for a given regular language L, it is always possible to construct an nfa with a single final state to accept L.
- Change the final state to initial state, initial state to final state and reverse the direction of edges in the transition graph. Then the new automaton accept L^R. So L^R is regular.

Homomorphism

- Suppose Σ and T are alphabets. A **homomorphism** $h(a)$ is a function
$$h : \Sigma \to T^*.$$
In other words, h substitutes a single letter in Σ by a string in T^*.
- The domain of a homomorphism can be extended to the set of strings by
$$h(w = a_1 \dots a_n) = h(a_1)h(a_2)\dots h(a_n).$$
- The **homomorphic image** of a language L is defined by
$$h(L) = \{h(w) : w \in L\}.$$

Closure under Homomorphism

- Let h be a homomorphism. If L is regular, then $h(L)$ is also regular.
- (proof) If L is regular, then $L = L(r)$ for some regular expression r. If we replace each symbol $a \in \Sigma$ of r by $h(a)$, the result $h(r)$ is a regular expression, which denotes $h(L)$. So $h(L)$ is regular.

Right Quotient

- The right quotient of L_1 with L_2 is defined by

$$L_1/L_2 = \{x : xy \in L_1 \text{ for some } y \in L_2\}.$$

- A string x is in L_1/L_2 even if there exist only one $y \in L_2$.
- To find L_1/L_2, we take any string in L_1. Each removal of a suffix in L_2 creates a string in L_1/L_2.

Closure under Right Quotient

- If L_1, L_2 are regular, then L_1/L_2, the right quotient of L_1 with L_2, is also regular.
- This can be shown by constructing a dfa for L_1/L_2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be the dfa that accepts L_1. We construct $\widehat{M} = (Q, \Sigma, \delta, q_0, \widehat{F})$ as follows.
  - For each $q_i \in Q$, construct $M_i = (Q, \Sigma, \delta, q_i, F)$.
  - Check if $L(M_i) \cap L(M_2)$ is empty (how?). If not, add q_i to $\widehat{F}$.

  Then $L(\widehat{M}) = L_1/L_2$.

Membership Question

- The **membership question** is

  Given a string w and a language L, is $w \in L$?

- A **membership algorithm** for a membership question must be able to give the correct answer for each instance of L and w.
- For a regular language L, a membership algorithm exists: We can simply run a dfa M for L on input w to see if M ends up in a final state.

Empty, Finite or Infinite

- For a regular language L, there exists algorithms for deciding whether L is empty, or whether L is infinite.
- To see this, let M be a dfa for L. Consider the transition graph of M.
  - If there is a simple path from q_0 to any $q \in F$, then L is not empty. Otherwise, L is empty.
  - To decide finiteness, find all vertices that are the base of a cycle. If any of these vertices is on a simple path from q_0 to a final state, then L is infinite. Otherwise it is finite.

Equality Question

- Given two regular languages L_1 and L_2, is $L_1 = L_2$?
- We can decide the answer of this question by defining

$$L_3 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2).$$

Note

$$L_3 = \emptyset \Leftrightarrow L_1 = L_2.$$

L_3 is regular by closure properties, so there exists an algorithm to decide if it is empty. If it is, then $L_1 = L_2$. Otherwise $L_1 \neq L_2$.

Non-Regular Languages

- Outside the regular set, there are many other languages.
- Since regular languages are associated with finite automata, if a language requires unlimited memory, then it cannot be regular. In particular,

  $L = \{a^n b^n : n \geq 0\}$ cannot be regular.

- Here we develop methods that can be used to prove a language to be non-regular.

The Pigeonhole Principle

- If we put n pigeons reside in m holes, with $n > m$, then there exists a hole that is occupied by at least two pigeons.
- We can use this principle to show L is non-regular.
- Suppose L is regular. Then there exists a dfa M for L. Consider $\delta_M^*(q_0, a^i)$. Since the number of states is finite, by the pigeonhole principle, there must be $m \neq n$ such that

$$\delta_M^*(q_0, a^m) = q = \delta_M^*(q_0, a^n).$$

  If $a^n b^n$ is accepted by M, then $a^m b^n$ is also accepted by M. This contradicts $L(M) = L$.

Pumping Lemma

- More generally, the pigeonhole principle can be used to derive **pumping lemmas**.
- A pumping lemma P of a set of languages S is a necessary condition for the membership of S.

$$L \in S \Rightarrow L \text{ satisfies } P.$$

- If L violates P, then L cannot be a member of S. This is how we prove a language to be non-regular: by showing that a pumping lemma for regular set is not satisfied.

Pumping Lemma for Regular Set

- Let L be an infinite regular language. There exists a positive integer m such that for any $w \in L$ with $|w| \geq m$ there exists x, y, z with

$$w = xyz, \text{ and } |xy| \leq m, \ |y| \geq 1,$$

such that

$$w_i = xy^i z \in L, \text{ for all } i = 0, 1, 2, \ldots.$$

- Any long-enough string in L can be decomposed into three parts. The middle part can be *pumped* any number of times and the resultant string is still in L.

Proof

- Let M be a dfa such that $L = L(M)$. Let $m = n + 1$ where $n + 1$ is the number of states. During the processing of a string w with $|w| = m$, $m + 1$ states have been visited so at least two of them are the same. Suppose the revisited state is q_r. We identify the label of the path from q_0 to the first q_r to be x, from q_r to q_r to be y, and from the last q_r to q_f to be z. Apparently, there is a cycle with base q_r. Repeating this cycle i times yields y^i. For any i, xy^iz corresponds to a path from q_0 to q_f so it is accepted by M.

Application of Pumping Lemma

- We use the pumping lemma to show that $L = \{a^n b^n : n \geq 0\}$ is not regular.
- Suppose L is regular, then an m as in the pumping lemma exists. To establish contradiction, we show that there exists a w with $|w| \geq m$ and there is no x, y, z to satisfy the required conditions.
- We choose $w = a^m b^m$. All decomposition falls into three categories: y contains a only, y contains b only and y contains both a and b. In any case, it is easy to show that $xy^i z$ cannot be in L for all i.

$\{ww^R : w \in \Sigma^*\}$

- As another example, we use the pumping lemma to show that $L = \{ww^R : w \in \{a,b\}^*\}$ is not regular.
- We use $w = a^m b^m b^m a^m$. Since $|xy| \leq m$, y can only contain a's. $xy^i z$ cannot be in L for all i since the run of a in the beginning is longer than in the end.
- The same idea can be used to prove $L = \{w \in \{a,b\}^* : n_a(w) < n_b(w)\}$ is not regular, by choosing $w = a^m b^{m+1}$.