Chapter 4

PROPERTIES OF REGULAR LANGUAGES

Learning Objectives

At the conclusion of the chapter, the student will be able to:

- State the closure properties applicable to regular languages
- Prove that regular languages are closed under union, concatenation, star-closure, complementation, and intersection
- Prove that regular languages are closed under reversal
- Describe a membership algorithm for regular languages
- Describe an algorithm to determine if a regular language is empty, finite, or infinite
- Describe an algorithm to determine if two regular languages are equal
- Apply the pumping lemma to show that a language is not regular

Closure Properties

- Theorem 4.1 states that if L_1 and L_2 are regular languages, so are the languages that result from the following operations:
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - L_1L_2
  - $\overline{L_1}$
  - L_1^*
- In other words, the family of regular languages is closed under union, intersection, concatenation, complementation, and star-closure.

Proof of the Closure Properties

- Since L_1 and L_2 are regular languages, there exist regular expressions r_1 and r_2 to describe L_1 and L_2, respectively
- The union of L_1 and L_2 can be denoted by the regular expression $r_1 + r_2$
- The concatenation of L_1 and L_2 can be denoted by the regular expression r_1r_2
- The star-closure of L_1 can be denoted by the regular expression r_1*
- Therefore, the union, concatenation, and star-closure of arbitrary regular languages are also regular

Proof of the Closure Properties (cont.)

- To prove closure under complementation of an arbitrary regular language L_1, assume the existence of a dfa M that accepts L_1
- A dfa M’ that accepts the complement of L_1 can be constructed as follows:
  - M’ has the same states, alphabet, transition function, and start state as M
  - The final states in M become non-final states in M’, while the non-final states in M become final states in M
- Since M’ accepts precisely the strings that M rejects, and M’ rejects precisely the strings that M accepts, then M’ accepts the complement of L_1, which is therefore shown to be regular

Proof of the Closure Properties (cont.)

- To prove that the intersection of two regular languages L_1 and L_2 is also regular, two basic approaches exist:
  - Given a dfa M_1 that accepts L_1 and a dfa M_2 that accepts L_2, construct a new dfa M' with states and transition function resulting from a combination of the states and transition functions from M_1 and M_2
  - Use DeMorgan's law to show that the intersection of L_1 and L_2 can be obtained by applying union and complementat

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

- Since the union and complementation operations have been shown to produce regular languages, the intersection of L_1 and L_2 must also produce a regular language

Closure under Reversal

- Theorem 4.2 states that if L is a regular language, so is L^R
- To prove closure under reversal, we can assume the existence of a nondeterministic finite automaton M with a single final state that accepts L
- Given the transition graph for M, to construct a nfa M^R that accepts L^R:
  - The start state in M becomes the final state in M^R
  - The final state in M becomes the start state in M^R
  - The direction of all transition edges in M is reversed

Elementary Questions about Regular Languages

- Given a regular language L and an arbitrary string w, is there an algorithm to determine whether or not w is in L?
- Given a regular language L is there an algorithm to determine if L is empty, finite, or infinite?
- Given two regular languages L_1 and L_2, is there an algorithm to determine whether or not $L_1 = L_2$?

A Membership Algorithm for Regular Languages

- Theorem 4.5 confirms the existence of a membership algorithm for regular languages
- To determine if an arbitrary string w is in a regular language L, we assume the existence of a standard unambiguous representation of L
- Given a standard representation of L, construct a dfa to accept L
- Simulate the operation of the dfa while processing w as the input string
- As previously stated, if the machine halts in a final state after processing w, then w is in L

Determining Whether a Regular Language is Empty, Finite, or Infinite

- Theorem 4.6 confirms the existence of an algorithm to determine if a regular language is empty, finite, or infinite
- Given the transition graph of a dfa that accepts L,
  - If there is a simple path from the start state to any final state, L is not empty (since it contains, at least, the corresponding string)
  - If a path from the start state to a final state includes a vertex which is the base of some cycle, L is infinite (otherwise, L is finite)

Determining Whether Two Regular Languages are Equal

- For finite languages, equality could be determined by performing a comparison of the individual strings
- More generally, theorem 4.7 confirms the existence of an algorithm to determine if two regular languages L_1 and L_2 are equal:
  - Define the language $L = (L_1 \cap L_2) \cup (L_1 \cap L_2)$
  - By closure, L is regular, so we can construct a dfa M to accept it, and by theorem 4.6, we can determine whether L is empty
  - L_1 and L_2 are equal if and only if L is empty

Identifying Nonregular Languages

- Although regular languages can be infinite, their associated automata have finite memory and are therefore incapable of accepting many languages
- To show that a language is not regular, two basic approaches exist:
  - Use the *pigeonhole principle* to construct a proof by contradiction
  - Use a *pumping lemma* for regular languages

Basis for the Pumping Lemma

- The transition graph for a regular language has certain properties:
  - If the graph has no cycles, the language is finite
  - If the graph has a nonempty cycle, the language is infinite
  - If the graph has such cycle, the cycle can either be skipped or repeated an arbitrary number of times, so if the cycle has label v and if the string w_1vw_2 is in the language, so are the strings w_1vvw_2, w_1vvvw_2, etc.
  - If such a cycle exists in a dfa with m states, the cycle must be entered by the time m symbols have been processed
- As a basis for the pumping lemma, we observe that given a language L, if any string in L does not satisfy these properties, L is not regular

A Pumping Lemma for Regular Languages

- Theorem 4.8: Given an infinite regular language L, every sufficiently long string *w* in L can be broken into three parts *xyz* such that
  - $|y| > 0$ and $|xy| \leq m$ (where m is an arbitrary integer $\leq |w|$)
  - An arbitrary number of repetitions of y yields another string in L
- The middle section, y, is said to be "pumped" to generate additional strings in L
- The pumping lemma can be used to show that, by contradiction, a certain language is not regular

Applying the Pumping Lemma to Show that a Language is not Regular

- The proof is similar to a game in which our goal is to show that a language L is not regular, while an opponent maintains the opposite:
    1. The opponent picks m
    2. We pick a string w in L so that $|w| \geq m$
    3. The opponent chooses the decomposition xyz, subject to $|y| > 0$ and $|xy| \leq m$, in a way that makes it hard to establish a contradiction
    4. We try to pick a number of repetitions i, such that xy^iz is not in L
- In general, we try to establish a strategy that allows us to show a contradiction regardless of the choices made in steps 1 and 3.