Chapter 4: Properties of Regular Languages

Dr. Yuan Yao

University of Nottingham Ningbo China (UNNC)

Learning Outcomes

Learning outcomes

At the conclusion of this chapter, the students are expected to 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.

Introduction

Review

- So far:
    - We have defined regular languages as those accepted by finite automata.
    - We have studied some ways in which they can be represented:
        - Finite Automata.
        - Regular Expressions.
        - Regular Grammars.
    - We have seen a few examples of their usefulness:
        - Pattern matching and search.
        - Specification of programming language constructs.
- Now, we will investigate some general properties of regular languages.

Properties of Regular Languages

- The followings are some general properties we may want to investigate:
  1. What happens when we perform operations (such as union, complementation, etc.) on regular languages?
     - For example, is it true that for every regular language L the complement $\overline{L}$ is also regular?
  2. Are there any algorithms that we can use to tell whether a given regular language is empty, non-empty, finite, or infinite?
     - For example, if a regular language L is given by a DFA M which accepts it, can we answer any of these questions just by inspecting M?
  3. Is every formal language regular? In other words, for any given $L \subseteq \Sigma^*$, is it possible to design a finite automaton M which accepts L?
     - You know the answer to this question already, but we have not provided a mathematical proof of the fact that $a^n b^n$ is not regular. In this chapter, we will.
     - More importantly, are there any methodical ways of telling whether a given language is regular or not?

Closure Properties

Closure Properties

- **Theorem 4.1:** If L_1 and L_2 are regular languages, then so are $L_1 \cup L_2$, $L_1 \cap L_2$, L_1L_2, $\overline{L_1}$, L_1^*. We say that the family of regular languages is closed under union, intersection, concatenation, complementation and star-closure.

- How to prove it?

Proof of the Closure Properties

- Suppose L_1 and L_2 are regular languages, then there existis regular expressions r_1 and r_2 such that $L_1 = L(r_1)$ and $L_2 = L(r_2)$.

- **It is not difficult to prove the closure properties for union, concatenation and start closure:**

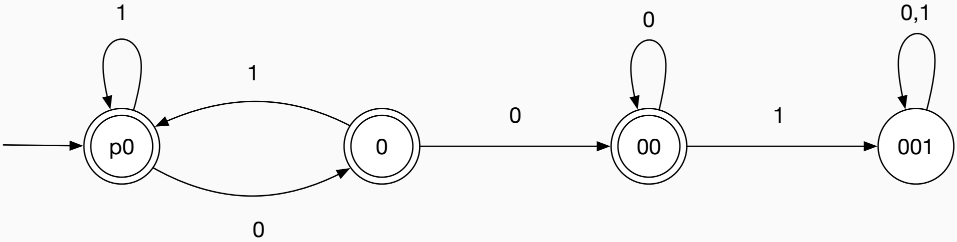
- Then it is obvious that $L_1 \cup L_2 = L(r_1) \cup L(r_2) = L(r_1 + r_2)$, as $r_1 + r_2$ is a regular expression, $L_1 \cup L_2$ is also regular.
- Also, $L_1L_2 = L(r_1)L(r_2) = L(r_1r_2)$, as r_1r_2 is a regular expression, L_1L_2 is therefore a regular language.
- Similarly, $L_1^* = L(r_1)^* = L(r_1^*)$, obviously L_1^* is a regular language.
- **Question:** How to prove complementation and intersection?

Proof of the Closure Properties

- Proof for the complementation $\overline{L}$:
    - **Hint:** remember how we construct a DFA that accpet the strings which does not have substring 001.

Proof of the Closure Properties

- Proof for the complementation $\overline{L}$:
  - **Hint:** remember how we construct a DFA that accpet the strings which does not have substring 001.



Proof of the Closure Properties

- Proof:
  - Assume there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts L_1.
  - We can construct another DFA $\overline{M}$ that accepts the complement of L_1 as follows:
    - $\overline{M}$ has the same states, alphabet, transition function, and start state as M.
    - The final states in M become non-final states in $\overline{M}$, while non-final states in M becomes final states in $\overline{M}$.
    - That is $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$
  - Since $\overline{M}$ accepts precisely the strings that M rejects, and $\overline{M}$ rejects precisely the strings that M accepts, then $\overline{M}$ accepts the complement of L_1.
  - Thus, $\overline{L_1}$ is regular.

Closure under Intersection

- To prove that the intersection of two regular languages L_1 and L_2 is also regular, two basic approaches exist:

- **An elegant logical approach:** Use DeMorgan's law to show that the intersection of L_1 and L_2 can be obtained by applying union and complementation:
$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$
- 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:
L_1 and L_2 are regular $\Rightarrow \overline{L_1}$ and $\overline{L_2}$ are regular $\Rightarrow \overline{L_1} \cup \overline{L_2}$ is regular $\Rightarrow \overline{\overline{L_1} \cup \overline{L_2}}$ is regular.

Closure under Intersection

- **A constructive approach:** Given a DFA $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ that accepts L_1 and a DFA $M_2 = (P, \Sigma, \delta_2, p_0, F_2)$ 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:

$$M' = (Q', \Sigma, \delta', (q_0, p_0), F')$$

- The state set Q' consists of pairs (q_i, p_j), where $q_i \in Q$ and $p_j \in P$

$$Q' = Q \times P$$

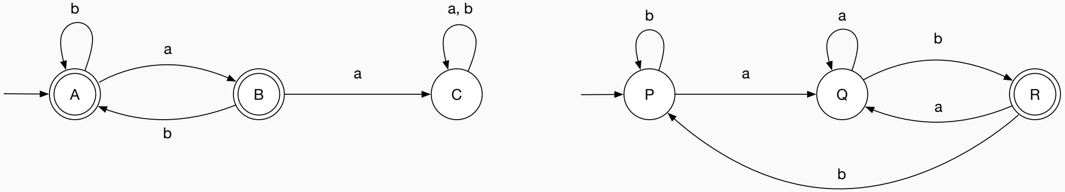
- The transition function δ' is such that M' is in state (q_i, p_j) whenever M_1 is in state q_i and M_2 is in state p_j.

$$\delta'((q_i, p_j), a) = (\delta_1(q_i, a), \delta_2(p_j, a))$$

- F' is defined as the set of all (q_i, p_j) such that $q_i \in F_1$ **and** $p_j \in F_2$.

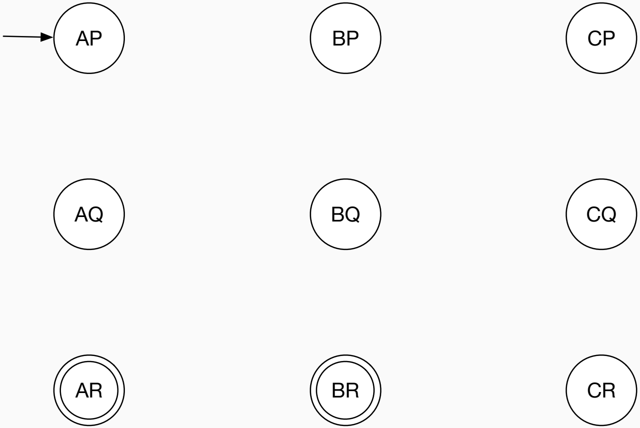
Example: Intersection of two languages

- Given the following two DFAs L_1 AND L_2 and their corresponding transition graphs, construct a DFA for the language $L_1 \cap L_2$.
  - $L_1 = \{w \in \{a, b\}^* \mid aa$ is not a substring of w $\}$
  - $L_2 = \{w \in \{a, b\}^* \mid w$ ends with $ab\}$



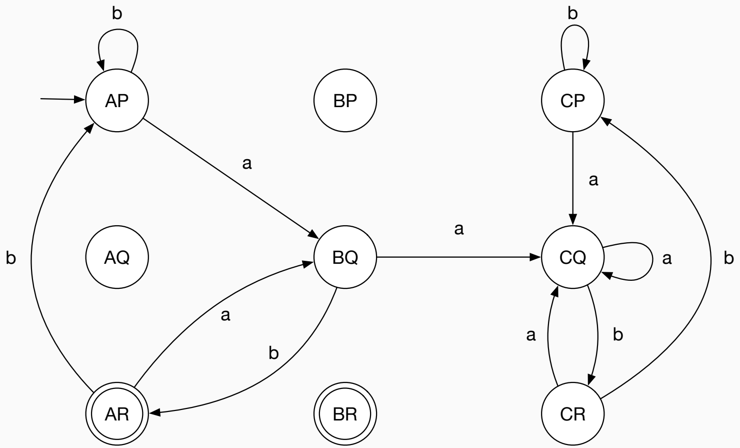
Example: Intersection of two languages

- Generate all combinations and label the initial state and final states.



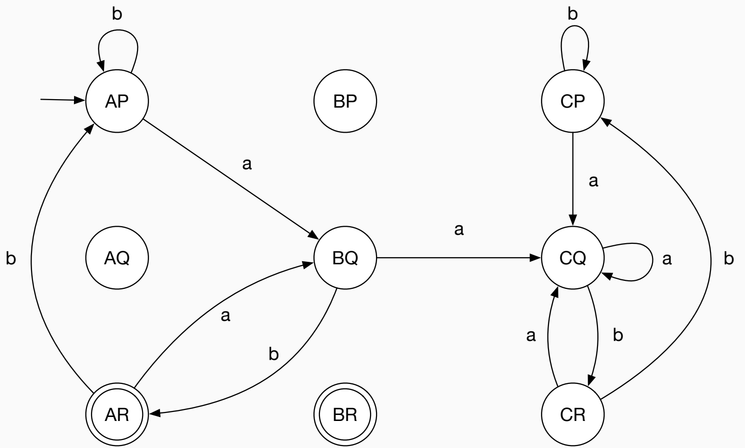
Example: Intersection of two languages

- Add the transitions between states.



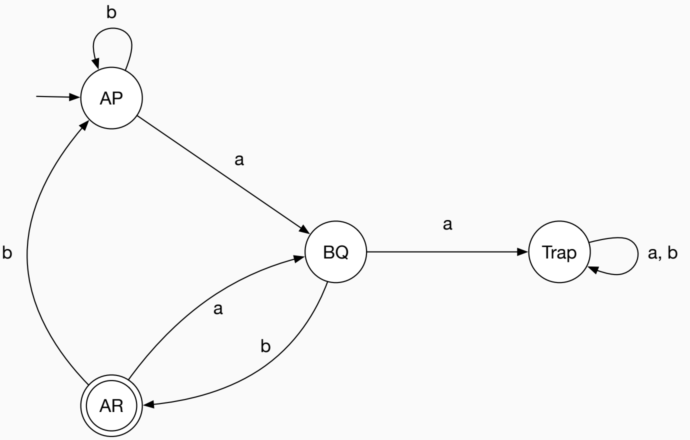
Example: Intersection of two languages

- How to improve it?



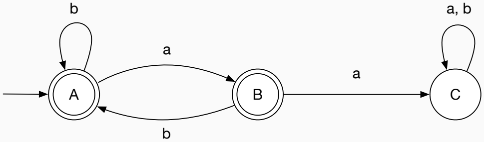
Example: Intersection of two languages

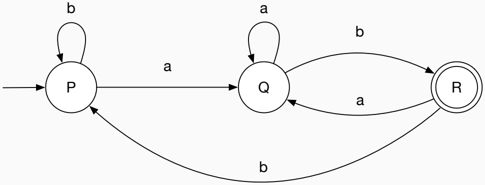
- Remove AQ, BP and BR.
- merge CP, CQ and CR to a single trap state.



Exercise

- Given the same DFAs L_1 and L_2
  - Construct a DFA which accepts $L_1 \cup L_2$.
  - Construct a DFA which accepts $L_1 - L_2$.





Closure under reversal

- **Theorem 4.2** 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 an NFA M^R that accepts L^R:
- How?

Closure under reversal

- **Theorem 4.2** 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 an 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 in M^R.

Exercise

- Given the fact that $L = \{ a^n b^m \mid n, m \geq 0\}$ is a regular language. Prove $\overline{L}$ is also a regular language.

Elementary Questions about Regular Languages

Elementary Questions

- The following are fundamental questions that should be asked about any class of formal languages:
    - Given a regular language L and an arbitrary string w, is there an algorithm to determine whether or not $w \in L$?
    - 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 $L_1 = L_2$?

Membership Algorithm

- We say that a regular language is given in a **standard representation** if and only if it is described by one of the following:
  - a finite automaton
  - a regular expression
  - a regular grammar

- **Theorem 4.5** Given a standard representation of any regular language L on L and $w \in \Sigma^*$, there exists an algorithm to determine whether or not $w \in L$.

- How to prove it?

Membership Algorithm

- We say that a regular language is given in a **standard representation** if and only if it is described by one of the following:
  - a finite automaton
  - a regular expression
  - a regular grammar

- **Theorem 4.5** Given a standard representation of any regular language L on L and $w \in \Sigma^*$, there exists an algorithm to determin whether or not $w \in L$.
- **Proof:** To determine if an arbitrary string w is in a regular language L, we assume we are given a standard representation of L, which we then convert to a DFA that accepts L.
- Simulate the operation of the DFA while processing w as the input string.
- If the machine halts in a final state after processing w, then $w \in L$, otherwise, $w \notin L$.

Empty, Finite or Infinite

- **Theorem 4.6** There exists an algorithm for determining whether a regular language, given in the standard representation, is empty, finite or infinite.
- How to prove it?

Empty, Finite or Infinite

- **Theorem 4.6** There exists an algorithm for determining whether a regular language, given in the standard representation, is empty, finite or infinite.

- The problem can be reduced to very simple and intuitive equivalent problems on graphs.

- 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).

Equivalence

- The question of the equality of two languages is not just of theoretical interest.
- It is also an important practical issue.

- For example, often several definitions of a construct in a programming language exist, and we need to know whether, in spite of their different appearances, they specify the same language.
- Even for regular languages the argument is not obvious.
- How to do it?

Equivalence

- For finite languages, equality can be determined by performing a comparison of the individual strings.
- However this cannot be done over infinite languages in finite time.
- Nor is it easy to see the answer by looking at the regular expressions, grammars, or DFAs.
- So, is it at all possible to test the equality of two regular languages in an algorithmic way?

Equivalence

- **Theorem 4.7** Given the standard representation of two regular languages L_1 and L_2, there exists an algorithm to determine whether or not $L_1 = L_2$.
- There is an elegant solution uses the already established closure properties.
    - Define the language $L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$
    - By closure properties, L is regular.
    - So, we construct a DFA M to accept it, and by Theorem 4.6, we can determine whether L is empty or not.
    - L_1 and L_2 are equal if and only if L is empty.

Identify Non-Regular Languages

Identifying Non-Regular Languages

- **Pigeonhole Principle:** If we put n objects into m boxes (pigeonholes), and if $n > m$, then at least one box must have more than one item in it.
- This simple principle is the basis of most of the methods for proving non-regularity of languages
- **Basic observation:** Although regular languages can be infinite, their associated automata have finite memory.

Prove A^nB^n is not regular: by Pigeonhole Principle

- We prove A^nB^n is not regular by directly applying the pigeonhole principle.
- Proof by contradiction: Assume that $A^nB^n = \{a^nb^n | n \geq 0\}$ is regular.
- Then, there must be a DFA $M = (Q, \{a, b\}, \delta, q_0, F)$ which accepts A^nB^n.
- Now, assume that Q has m elements, i.e., the DFA has m states, and consider the set:

$$X = \{a, a^2, a^3, \ldots, a^m, a^{m+1}\}$$

- Then, consider the set of states:

$$Y = \{\delta^*(q_0, a), \delta^*(q_0, a^2), \ldots, \delta^*(q_0, a^m), \delta^*(q_0, a^{m+1})\}$$

- By the pigeonhole principle, for some $i \neq j$ we must have:

$$\delta^*(q_0, a^i) = \delta^*(q_0, a^j)$$

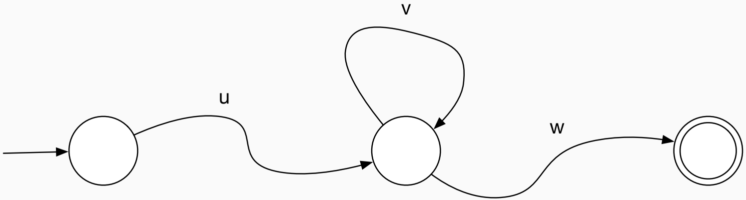
Prove A^nB^n is not regular: by Pigeonhole Principle

- So, we can assume for some $q \in Q$, we have $\delta^*(q_0, a^i) = \delta^*(q_0, a^j) = q$
- As $a^ib^i \in A^nB^n$, we must have $\delta^*(q, b^i) \in F$.
- However, we also had $\delta^*(q_0, a^j) = q$, which, together with $\delta^*(q, b^i) \in F$, implies that $a^jb^i \in A^nB^n$, which is a contradiction because $i \neq j$.
- Therefore, there cannot be any DFA accepting A^nB^n, and the language is not regular.

The Pumping Lemma

- Suppose that $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA with n states that accepts a language L:
    - If it accepts a string x such that $|x| \geq n$, then, by the time n symbols have been read, M must have entered some state more than once.
        - Again, this is a consequence of the pigeonhole principle.
    - In other words, there must be two different prefixes u and uv such that:

$$\delta^*(q_0, u) = \delta^*(q_0, uv)$$



The Pumping Lemma

- This implies that there are many more strings in L, because we can traverse the loop v any number of times (including leaving it out altogether).
- In other words, all of the strings uv^iw for $i \geq 0$ are in L.
- This fact is known as the **Pumping Lemma for Regular Languages**.

The Pumping Lemma

- **Theorem:** Suppose that L is a language over Σ. If L is accepted by the DFA $M = (Q, \Sigma, \delta, q_0, F)$, then there is an integer n so that for every x in satisfying $|x| \geq n$, there are three strings u, v, and w such that $x = uvw$ and:
  - $|uv| \leq n$
  - $|v| > 0$, i.e., $v \neq \lambda$
  - For every $i \geq 0$, the string uv^iw belongs to L.
- The way we found n was to take the number of states in an FA accepting L.
- In typical applications, we do not need to know this, only that there exists such an n.

The Pumping Lemma

- The statement of the Pumping lemma is as follows:
  - If L is regular $\Rightarrow$ the pumping property holds.
- In practice, we use the contrapositive:
  - If the pumping property does not hold $\Rightarrow$ the language L is not regular.
- Thus, the most common application of the pumping lemma is to show that a language is not regular.
- The proof is by contradiction. We suppose that the language can be accepted by an FA, and we let n be the integer in the pumping lemma.
- Then we choose a string x with $|x| \geq n$ to which we can apply the lemma to get a contradiction.

Prove A^nB^n is not regular: by Pumping Lemma

- Let us prove, using the Pumping lemma, that $L = A^nB^n$ cannot be accepted by an FA.
- Suppose, for the sake of contradiction, that A^nB^n is accepted by an FA; let n be as in the pumping lemma;
- Choose $x = a^nb^n$, then $x \in A^nB^n$ and $|x| \geq n$;
- Therefore, by the Pumping lemma, there are strings u, v and w such that $x = uvw$ and the 3 conditions hold;
- Because $|uv| \leq n$ and x starts with n $a's$, all the symbols in u and v are $a's$; Therefore, $v = a^k$, for some $k > 0$;
- The pumping property implies that $uvvw \in A^nB^n$, so $a^{n+k}b^n \in A^nB^n$.
- This is a contradiction, and we conclude that A^nB^n cannot be accepted by an FA. So, A^nB^n is not regular.

Exercise

- Prove that $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$ is not regular.

The Pumping Lemma

- There are other languages that are not accepted by any FA, among them:
  - Balanced, the set of balanced strings of parentheses;
  - Expr, the language of simple algebraic expressions;
  - The set L of legal C programs.
- In all three examples, because of the nature of these languages, a proof using the pumping lemma might look a lot like the proof for $A^n B^n$ in our first example.

- Note that the Pumping lemma only provides a necessary condition, but not a sufficient one.
  - If L is regular $\Rightarrow$ the pumping property holds
- In other words, it does not say anything about the pumping property when L is not regular.
- In fact, there are languages that do satisfy the pumping lemma, but are not regular, e.g.:
  $L = \{a^i b^j c^j \mid i \geq 1 \text{ and } j \geq 0\} \cup \{b^j c^k \mid j \geq 0 \text{ and } k \geq 0\}$.