Theory of Automata & Formal Languages
Mr. Saurabh Singh

Pumping Lemma in Theory of Computation:

There are two pumping introductions and their definition is

1. Regular language, and

2. Context-free language

Pumping Lemma for Regular Languages:

Common language abstraction

For any regular language L, there is an integer n such that for all x ∈ L ≥ n with | x |, there are u, v, w∈∑ * such that x = uvw, and

(1) | uv | ≤n

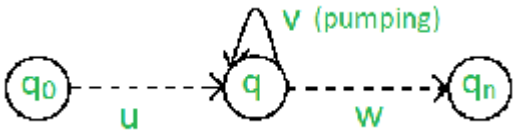
(2) | v | ≥1

(3) For all i≥0: $uv^iw \in L$

In short, this means that if the string v is "extracted", that is, if v is inserted any number of times, the resulting string remains in L.

Cited quotes are used as proof of language irregularities. Therefore, if a language is regular, it can always satisfy the Tic Lemma. If at least one string made of sucker is not in L, then L must be irregular.

The opposite may not always be true. In other words, if Pumping Lemma is established, it does not mean that the language is formal.



For example, let us prove that L01 = $\{0^n1^n \mid n\geq 0\}$ is irregular.

Let us assume that L is regular, then by drawing arguments, we can follow the given rules above.

Now, let x ∈ L and | x | ≥ n. Therefore, by drawing arguments, there are u, v, w, so that (1)-(3) holds.

We prove that (1)–(3) does not hold for all u, v, w.

If (1) and (2) are true, then x = 0n1n = uvw, |uv | ≤n and | v | ≥1.

Therefore, u = 0a, v = 0b, w = 0c1n where: a + b ≤ n, b≥1, c≥0, a + b + c = n

But then (3) fails for i = 0

$uv^0w = uw = 0^a0^c1^n = 0^{a+c}1^n \notin L$, because a + c ≠ n.

Theory of Automata & Formal Languages
Mr. Saurabh Singh

Closure properties of Regular languages:

The closure property of a daily language is defined as certain operations on the regular language, and these operations can guarantee the assembly of the regular language. Closure refers to a particular operation on a particular language, leading to a replacement language having an equivalent "type" because the original language (i.e., regular language).

The regular language is closed under the subsequent operations.

Consider L and M are regular languages:

1. Kleen closed: RS may be a regular expression whose language is L and M. R * may be a regular expression with language L *.

2. Closed: RS may be a regular expression whose language is L and M. R^+ may be a Regular expression whose language is L^+.

3. Complement:

The complement of language L (for the letter E, such that E ^ * contains L) is E ^ *-L. Since E^* is definitely regular, the complement of regular language is always regular.

4. Reverse operator:

In the case of a given language L, L^R is a set of character strings which is reversed to L.

Example: L = {0, 01, 100};

L ^ R = {0, 10, 001}.

Proof: Let E be a regular expression of L. We demonstrate how to reverse E to provide a regular expression E^R for L^R.

5. Complement:

The complement of language L (for the letter E, such that E ^ * contains L) is E ^ *-L. Since E^* is definitely regular, the complement of regular language is always regular.

6. Alliance:

Let L and M be the language of regular expressions R and S, respectively, and then R + S is the regular expression whose language is (L U M).

7. Intersection:

Assuming that L and M are the languages of regular expressions R and S, respectively, it is a regular expression whose language is the intersection of L and M.

Proof: Assume that A and B are DFA and their languages are L and M respectively. Constructing C, the product automaton of A and B makes the final state of C a pair consisting of the final states of A and B.

8. Set the difference operator:

If L and M are regular languages, then L-M = string in L, but not M.

Proof: Assume that A and B are DFA and their languages are L and M respectively. Constructing C, the product automaton of A and B makes the final state of C a pair, where the A state is the final state and the B state is not.

9. Homomorphism:

Homomorphism on the alphabet is a function that provides a string for each symbol in the alphabet. Example: h (0) = ab; h (1) = E. Expand to a string by h (a1...an) = h (a1)... h (an). For example: h (01010) = abababab.

If L may be a regular language, and h may be a homomorphism of its letters, then h(L) = {h(w)| w in L} is additionally a daily language .|If L may be a regular language, and h may be a homomorphism of its letters, then h(L) = {h(w)| w in L} is additionally a daily language.

Proof: Let E be a regular expression of L. Apply h to every symbol in E. As a result, the language E of R is h (L).

10. Inverse homomorphism: Let h be a homomorphism, L be an output language whose letter is h. h ^ -1(L) = {w | h (w) in L}.

Note: Under the closure properties of regular languages, more properties such as symmetric difference operators, prefix operators, and substitutions are also turned off.

Decision attributes:

In the case of finite automata, almost all attributes are determinable.

i. Emptiness
ii. Not empty
iii. Limited
iv. Unlimited
v. Membership
vi. Equality

These explanations are as follows.

(I) Emptiness and non-emptiness:

Step 1: Select the unreachable state from the initial state, and then delete it (delete the unreachable state).

Step 2: If the generated machine contains at least one final state, the finite automaton will accept non-empty languages.

Step 3: If the generated machine has no final state, the finite automaton accepts the empty language.

(II) Finiteness and Infinity:

Step 1: Select the inaccessible state from the initial state, and then delete it (delete the inaccessible state).

Step 2: Select the state that cannot reach the final state and delete (delete the invalid state).

Step 3: If the generated machine contains loops or loops, the finite automata accepts unlimited languages.

Step 4: If the generated machine does not contain loops or loops, the finite automaton will accept unlimited languages.

(III) Membership:

Membership is an attribute used to verify whether a finite automaton accepts any string, that is, whether it is a member of the language.

Let M be a finite automaton that accepts some character strings on letters, and 'w' is any character string defined on letters, if there is a transition path in M, then the transition path starts with the initial state and ends with any final The state is over, then the string 'w' is a member of M, otherwise 'w' is not a member of M.

(IV) Equality:

Two finite state automata M1 and M2 are equal if and only if they accept the same language. Minimize the finite state automata, the smallest DFA will be unique.