Theory of Computation

Shukla Banik

Department of Computer Science & Engineering
Siksha 'O' Anushandhan (Deemed to be University)

shuklabanik@soa.ac.in

October 12, 2020

The Regular Operations

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

- **Union**: $A \cup B = \{x| \, x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy| \, x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1x_2 \ldots x_k| \, k \geq 0 \text{ and each } x_i \in A\}$.

The union operation simply takes all the strings in both A and B and lumps them together into one language. The concatenation operation is a little trickier. It attaches a string from A in front of a string from B in all possible ways to get the strings in the new language. The star operation is a unary operation instead of a binary operation, which applies to a single language rather than to two different languages. It works by attaching any number of strings in A together to get a string in the new language. "Any number" includes 0 as a possibility, the empty string ϵ is always a member of A^*, no matter what A is.

The Regular Operations

Example 1: Let the alphabet Σ be the standard 26 letters $\{a, b, ..., z\}$. If $A = \{good, bad\}$ and $B = \{boy, girl\}$, then

$$A \cup B = \{\texttt{good}, \texttt{bad}, \texttt{boy}, \texttt{girl}\},$$

$$A \circ B = \{\texttt{goodboy}, \texttt{goodgirl}, \texttt{badboy}, \texttt{badgirl}\}, \text{ and}$$

$$A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad}, \ldots\}.$$

■

Let $N = \{1, 2, 3, ...\}$ be the set of natural numbers. When we say that N is closed under multiplication, we mean that for any x and y in N , the product $x \times y$ also is in N . In contrast, N is not closed under division, as 1 and 2 are in N but $1/2$ is not. Generally speaking, a collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.

The Regular Operations

Theorem 1: The class of regular languages is closed under the union operation. In other words, if A1 and A2 are regular languages, so is A1 ∪ A2.

Proof Idea: We have regular languages A_1 and A_1 and want to show that $A_1 \cup A_2$ also is regular. Because A_1 and A_2 are regular, we know that some finite automaton M_1 recognizes A_1 and some finite automaton M_2 recognizes A_2. To prove that $A_1 \cup A_2$ is regular, we demonstrate a finite automaton, call it M , that recognizes $A_1 \cup A_2$.
This is a proof by construction. We construct M from M_1 and M_2. Machine M must accept its input exactly when either M_1 or M_2 would accept it in order to recognize the union language. It works by simulating both M_1 and M_2 and accepting if either of the simulations accept. It first simulates M_1 on the input and then simulates M_2 on the input. Once the symbols of the input have been read and used to simulate M_1, we can't "rewind the input tape" to try the simulation on M_2. That way, only one pass through the input is necessary.

The Regular Operations

All we need to remember a pair of states that each machine would be in. If M_1 has k_1 states and M_2 has k_2 states, the number of pairs of states, one from M_1 and the other from M_2, is the product $k_1 \times k_2$. This product will be the number of states in M, one for each pair. The transitions of M go from pair to pair, updating the current state for both M_1 and M_2. The accept states of M are those pairs wherein either M_1 or M_2 is in an accept state.

The Regular Operations

Theorem 1: The class of regular languages is closed under the union operation. In other words, if A1 and A2 are regular languages, so is A1 $\cup$ A2.

Proof: Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. We have to construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$. Q is the cartesian product of sets Q_1 and Q_2 and is written $Q_1 \times Q_2$. It is the set of all pairs of states, the first from Q_1 and the second from Q_2.
2. Σ, the alphabet, is the same as in M_1 and M_2. In this theorem and in all subsequent similar theorems, we assume that both M_1 and M_2 have the same input alphabet Σ. The theorem remains true if they have different alphabets, Σ_1 and Σ_2. We would then modify the proof to $\Sigma = \Sigma_1 \cup \Sigma_2$.

The Regular Operations

3. δ, is the transition function, is defined as follows.
   For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$,
   $\delta'((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$. Hence δ gets a state of M (which actually is a pair of states from M_1 and M_2), together with an input symbol, and returns $M's$ next state.
4. q_0 is the pair (q_1, q_2).
5. F is the set of pairs in which either member is an accept state of M_1 or M_2. We can write it as
   $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

   This concludes the construction of the finite automaton M that recognizes the union of A_1 and A_2. We have just shown that the union of two regular languages is regular, thereby proving that the class of regular languages is closed under the union operation.

We now turn to the concatenation operation and attempt to show that the class of regular languages is closed under that operation, too.

Theorem 2: The class of regular languages is closed under the concatenation operation.

To prove this theorem, let's try something along the lines of the proof of the union case. As before, we can start with finite automata M1 and M2 recognizing the regular languages A_1 and A_2. But now, instead of constructing automaton M to accept its input if either M_1 or M_2 accept, it must accept if its input can be broken into two pieces, where M_1 accepts the first piece and M_2 accepts the second piece.

Nondeterministic Finite Automaton (NFA)

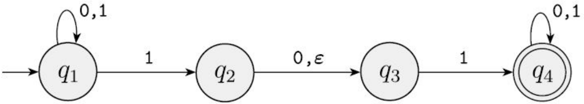
When the machine is in a given state and reads the next input symbol, we know what the next state will be - it is determined. We call this deterministic computation. In a nondeterministic machine, several choices may exist for the next state at any point.
In a nondeterministic finite automaton (NFA), for each state there can be zero, one, two, or more transitions corresponding to a particular symbol. If NFA gets to state with more than one possible transition corresponding to the input symbol, we say it branches. If NFA gets to a state where there is no valid transition, then that branch dies. An NFA accepts the input string if there exists some choice of transitions that leads to ending in an accept state. Thus, one accepting branch is enough for the overall NFA to accept, but every branch must reject for the overall NFA to reject.

Nondeterministic Finite Automaton

Formal Definition

A nondeterministic finite automaton is a 5-tuple (Q, Σ, δ, q0, F), where

1. Q is a finite set of states.
2. Σ is a finite alphabet.
3. δ: Q $\times$ Σ_ϵ $\rightarrow$P(Q) is the transition function.
4. q0 $\in$ Q is the start state.
5. F $\subseteq$ Q is the set of accept states.



Example 2: Write down the formal definition of the above NFA N .

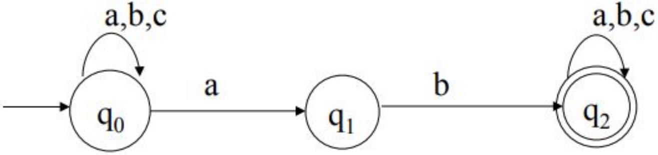
The formal description of NFA N is (Q, Σ, δ, q0, F), where

1. Q = {q1, q2, q3, q4}
2. Σ = {0,1}
3. δ:

States	0	1	ϵ
q1	{q1}	{q1,q2}	ϕ
q2	{q3}	ϕ	{q3}
q3	ϕ	{q4}	ϕ
q4	{q4}	{q4}	ϕ

4. q0 = {q1} is the start state
5. F = {q4}

Example 3: Let $\Sigma = \{a, b, c\}$. Draw an NFA that accepts all strings which contains the substring ab.



Example 3: Let $\Sigma = \{a, b\}$. Draw an NFA that accepts all strings where the third to the last symbol in the string is b.