Conversion of Regular Expression to DFA

We know the concept of **deterministic finite automata** (DFA) from the very basics of automata theory. We also learnt the concept of **regular expressions** and their properties. In this chapter, you will learn how to convert a given regular expression to its equivalent finite automata.

Conversion Rules of Regular Expressions to Finite Automata

Here we will highlight some basic rules of regular expressions that we use in different expressions. Let us see the rules and corresponding DFA drawing.

Expression: a + b (a union b)

If you have an expression of the form "A + B", you need two states. Let's call them state A and state B. The state A will transition to state B upon receiving input 'a'. Similarly, state A will transition to state B upon receiving input 'b'.

Alternatively, you can represent it using one transition line for both inputs. This means that the state will transition to the next state upon receiving either input 'a' or input 'b'.

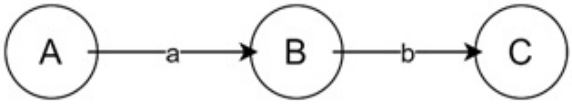
Expression: ab (a followed by b)

When dealing with expressions of the form AB, you need three states: A, B, and C. State A transitions to state B on input 'a', and state B transitions to state C on input 'b'.

Unlike the previous case, where the transition to the next state could be triggered by either input, here the transitions are sequential. State A will transition to state B only on input 'a', and state B will transition to state C only on input 'b'.

So, for expressions of this form, you need separate transitions for each input –

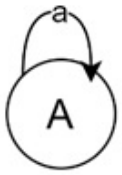
- State A transitions to state B on input 'a'.
- State B transitions to state C on input 'b'.



Expression: a* (closure of a)

For expressions involving the closure of A, represented as A*, the representation is straightforward. You create a state that loops back to itself on input 'a'. This means the state can accept any number of 'a's, including zero.

To summarize, for A* − State A transitions to itself on input 'a', allowing any number of 'a's.



These are three essential rules to remember when designing finite automata from given regular expressions.

Examples of Regular Expression to DFA

We have seen the rules. Now we will use these rules to solve the expressions and convert them to finite state diagrams

Example 1: B a* b

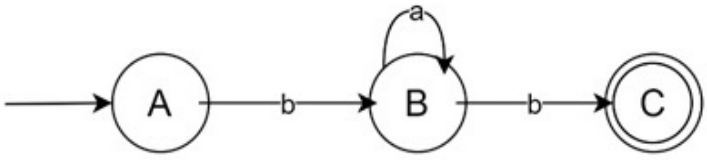
The strings accepted by this regular expression are of the form B, followed by zero or more A's, and ending with a B. For instance, BB, BAB, BAAB, and so on.

To design the finite automata −

- Start with state A (initial state).
- State A transitions to state B on input 'b'.
- State B has a self-loop on input 'a' (closure).
- State B transitions to state C on another input 'b' (final state).

So, for **B A* B** −

- State A transitions to state B on input 'b'.
- State B transitions to itself on input 'a'.
- State B transitions to state C on input 'b'.



This design will accept strings like BB, BAB, BAAB, etc.

Example 2: a + b followed by c

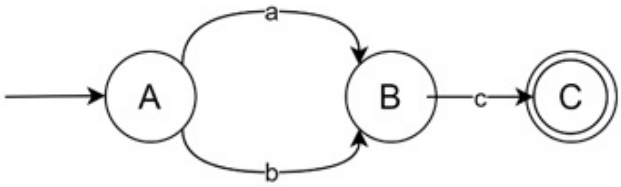
Here, we have a union operation followed by an 'and' operation. The regular expression is A + B followed by C. This means A or B, followed by C.

To design the finite automata –

- Start with state A (initial state).
- State A transitions to state B on input 'a'.
- State A also transitions to state B on input 'b'.
- State B transitions to state C on input 'c' (final state).

So, **for A + B followed by C** –

- State A transitions to state B on input 'a' or 'b'.
- State B transitions to state C on input 'c'.



This design will accept strings like AC and BC.

Example 3: a (bc)*

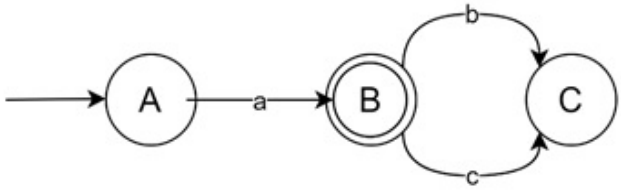
In this expression, A is followed by zero or more repetitions of the sequence BC.

To design the finite automata –

- Start with state A (initial state).
- State A transitions to state B on input 'a'.
- State B transitions to state C on input 'b'.
- State C transitions back to state B on input 'c'.
- State B (also final state) ensures that the string is accepted when ending with C.

So, for **A (BC)*** –

- State A transitions to state B on input 'a'.
- State B transitions to state C on input 'b'.
- State C transitions back to state B on input 'c'.

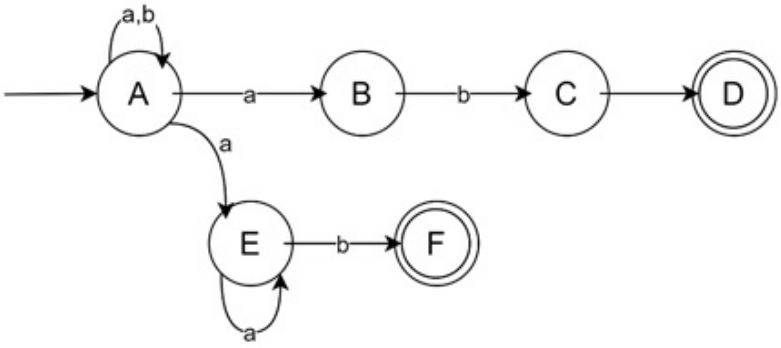


This design will accept strings like A, ABC, ABBC, ABBCC, and so on.

Example 4: (a | b)*(abb| a + b)

This is little complex expression. Here we can divide the whole expression into separate parts. **(a | b)*** and **(abb | a + b)** are concatenated together, as we can see.

From the previous knowledge, we can draw the diagram directly which will be similar like the one shown below –



Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Conclusion

In this chapter, you learnt how to convert regular expressions to finite automata by following some simple strategies. By understanding and applying these rules, you can convert regular expressions to their equivalent finite automata.

We highlighted three primary forms: the **A + B** (**union**), **AB** (**concatenation**), and the **A*** (**closure**). These rules form the foundation for designing finite automata from regular expressions.