Convert Regular Expression to Regular Grammar

In this chapter, we will cover the process of converting a regular expression to a regular grammar. As we know a regular expression is a sequence of characters that defines a search pattern. It is used in many applications including text editors, search engines, and programming languages.

A regular grammar, in contrast, is a set of production rules that define the syntax of a regular language. The production rules of a regular grammar can be used to generate strings in the language. Here we will see the rules and with examples in detail.

Regular Expression

A **regular expression** is a sequence of characters that specifies a search pattern. Regular expressions are used in many applications, including text editors, search engines, and programming languages. They are used to find and replace text, validate data, and parse data.

Regular Grammar

A regular grammar is a formal grammar that generates a regular language. It is a type 3 grammar in the Chomsky hierarchy. This type of grammar is used to represent the syntactic structure of a regular language. It defines the rules for generating strings in the language.

The rules of a regular grammar can be expressed in the following format −

$$\mathrm{A} \rightarrow \mathrm{aB}$$

$$\mathrm{A} \rightarrow \mathrm{a}$$

Converting a Regular Expression to a Regular Grammar

Let us see the steps involved to convert the regular expressions to regular grammars.

The process of converting a regular expression to a regular grammar is simple. It is constructing an intermediate nondeterministic finite automaton (NFA) with epsilon transitions. As we know, the Epsilon transitions allow the automaton to change states without consuming any input symbols.

Let us see a step-by-step procedure −

Step 1: Construct NFA with Epsilon Transitions

The first step is to convert the given regular expression into an equivalent NFA. This NFA can include epsilon transitions. This step essentially visualizes the regular expression as a state machine.

Step 2: Eliminate Epsilon Transitions

Once the NFA is constructed, we need to eliminate all the epsilon transitions. This results in an equivalent deterministic finite automaton (DFA). This step ensures that each transition in the state machine corresponds to a specific input symbol −

Step 3: Convert DFA to Regular Grammar

The final step is to convert the DFA into a regular grammar. The procedure for this step is straightforward.

- The states in the DFA become the non-terminal symbols in the regular grammar.
- The transitions in the DFA become the production rules in the regular grammar

Example for Converting Regular Expression to a Regular Grammar

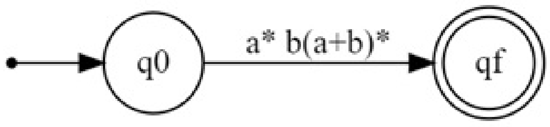
Let us see an example to convert regular example to regular grammar as we discussed through steps.

Consider we have the following regular expression −

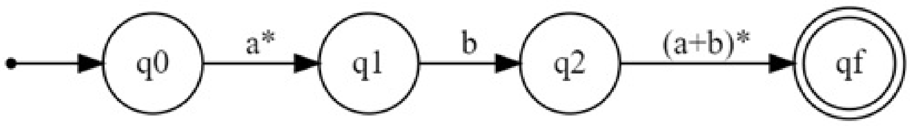
$$a^* \, b \, (a + b)^*$$

Construct NFA with Epsilon Transitions at first, here we begin by constructing an NFA with epsilon transitions that accepts the language described by the regular expression.

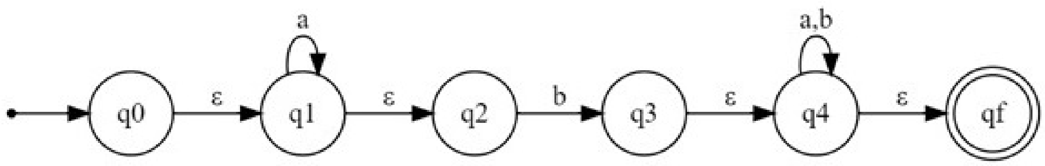
Let us see the generation of finite automata in steps.



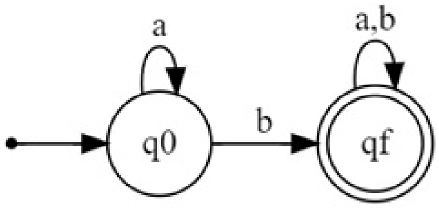
Now break the steps,



Now break this to epsilon nfa −



Now **construct the DFA** −



Next the most important and interesting part that is converting this DFA to a regular grammar.

The machine description is like below −

$$Q = \{q0,\ qf\}$$

$$\Sigma = \{a,\ b\}$$

$$\delta = \{(q0,\ a) \rightarrow q0,\ (q0,\ b) \rightarrow qf,\ (qf,\ a) \rightarrow qf,\ (qf,\ b) \rightarrow qf\}$$

$$q0\ \text{ is initial state}$$

$$F = \{qf\}$$

We know that a regular grammar is four tuple. **G = {V, T, P, S}**

```
V = {q0, qf}
T = {a, b}
S = {q0}
P = {
   q0 → aq0,
   q0 → bqf,
   q0 → b
   qf → aqf
   qf → a
   qf → bqf
   qf → b
}
```

Here we can see it can also a string "aaaab" since (a + b)* is there, it will be null or more for (a + b) so for that reason it is using q0 → b transition. Similar is happening for qf → a and qf → b.

Conclusion

The process of converting a regular expression to a regular grammar is simple. In this chapter, we explained the steps in detail along with an example where we demonstrated how to convert a given regular expression into segmented finite automata and then finally removing epsilon transition to get the DFA. Thereafter, we converted that to grammar itself.