Equivalence of Two Regular Expressions in Automata

Regular Expressions (REs) are used in defining patterns for string manipulation. Sometimes we have multiple regular expressions and we need to check their equivalence. Understanding the equivalence of REs is needed for designing efficient and reliable algorithms for pattern recognition and language processing.

In this chapter, we will focus on the concept of equivalence, exploring its meaning and the various methods to determine whether two given REs are equivalent or not.

Equivalence of Regular Expressions in Automata

Equivalence of Regular Expressions means the situation where two different REs represent the same set of strings. In simpler words, if two REs generate the same language, they are considered equivalent.

Checking whether two REs are equivalent is a fundamental problem in the theory of computation. Let us see the methods for checking.

Methods for Proving Equivalence

There are primarily three methods for proving the equivalence of two REs −

- Set Generation
- Identities
- Machine Equivalence

Let's understand each of these methods in detail.

Set Generation

This method involves generating the set of strings accepted by each RE and then comparing the two sets. If the sets are identical, the REs are equivalent. Let us understand this through an example −

- If we have RE1 = "1" and RE2 = "2," we can generate the set of strings accepted by each. Using RE1, the set would be {1}, and using RE2, the set would be {2}. Since these sets are different, we can conclude that RE1 and RE2 are not equivalent.
- On the other hand, if RE1 = "1" and RE2 = "1," the generated sets would both be {1}. Therefore, RE1 and RE2 are considered equivalent in this case.

Identities

This method uses a set of established identities that hold true for the regular expressions. These identities allow us to manipulate and simplify expressions. Ultimately checking whether the two expressions can be reduced to the same form or not.

For instance, one identity states that **"ε + R* = R"** where Epsilon represents the empty string. If we have **RE1 = "ε + R"** and **RE2 = "R"** we can use this identity to simplify RE1 to **"R"** which matches RE2. Consequently, we can say that RE1 and RE2 are equivalent.

Machine Equivalence

This method involves constructing finite automata (FA) for each RE and then checking whether the two FAs are equivalent or not. If the FAs accept the same set of strings, then the REs they represent are equivalent.

The machine equivalence method is typically used for more complex REs where set generation and identity application become problematic.

Examples to Prove Equivalence with Identities

Example 1

Let us consider an example to explain the use of identities for proving RE equivalence. Suppose we want to prove that −

$$(1\,+\,(00)^*\,1)\,+\,(1\,+\,(00)^*\,1)\,(0\,+\,(10)^*\,1)^*\,(0\,+\,(10)^*\,1)\,=\,0^*\,(1(0\,+\,(10)^*\,1))^*$$

We can start by simplifying the left-hand side (LHS) of the equation using the identities −

The LHS is $(1 + (00)^* 1) + (1 + (00)^* 1)\ (0 + (10)^* 1)^*\ (0 + (10)^* 1)$

Let us take $(1 + 00^* 1)$ as a common factor, so

$$(1\,+\,(00)^*\,1)(\varepsilon\,+\,(0\,+\,(10)^*1)^*(0\,+\,(10)^*1))$$

$$(\varepsilon\,+\,\mathrm{R}^*\mathrm{R}),\ \text{where}\,\mathrm{R}\,=\,(0\,+\,(10)^*\,1)$$

As we know,

$$(\varepsilon\,+\,\mathrm{R}^*\,\mathrm{R})\,=\,(\varepsilon\,+\,\mathrm{R}\mathrm{R}^*)\,=\,\mathrm{R}^*$$

$$(1\,+\,(00)^*1)((0\,+\,(10)^*>1)^*)$$

out of this consider,

$$(1\ +\ (00)^*1)(0\ +\ (10)^*1)^*$$

Taking 1 as a common factor, we can get

$$(\varepsilon\ +\ (00)^*)1(0\ +\ (10)^*1)^*$$

Applying ε + $(00)^*$ = 0^*, we get

$$0^*1(0\ +\ (10)^*1)^*$$

Which is the R.H.S.

Let us see another example using set equalization method −

Example 2

Prove that $(0^*\ 1^*)^*$ = $(0 + 1)^*$

The LHS is $(0^*\ 1^*)^*$

$$(0^*1^*)^*\ =\ \{\varepsilon,\ 0,\ 00,\ 1,\ 11,\ 111,\ 01,\ 10,\ \ldots\}$$

This is any combination of 0's, any combination of 1's, any combination of 0 and 1, ε

On the RHS, $(0 + 1)^*$

$$(0\ +\ 1)^*\ =\ \{\varepsilon,\ 0,\ 00,\ 1,\ 11,\ 111,\ 01,\ 10,\ \ldots\}$$

This is also any possible combination of 0s and 1s including null.

So, the sets are similar and these are equivalent.

Conclusion

In this chapter, we explained the concept of equivalence of regular expressions and described three methods to prove it.

Generating strings can be tedious; identities make the process easier for complex regular expressions. Constructing and comparing finite automata offers an automated approach but can be computationally expensive. The choice of method depends on the complexity of the regular expressions and the resources available.