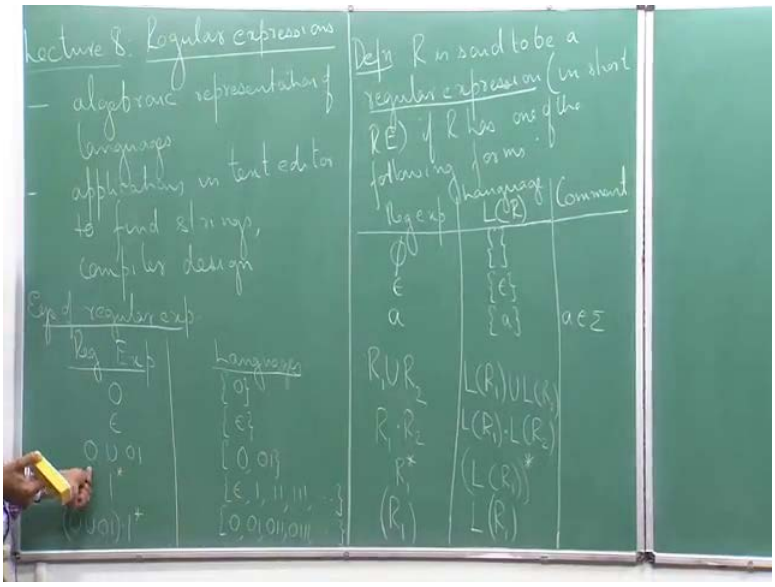
Theory of Computation
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 08
Regular Expressions

Welcome to the 8th lecture of this course. Today, we will discuss another way of representing languages namely that of a regular expression. So, regular expressions are an algebraic way to represent languages; and as we shall see that the classes of languages that can be expressed via regular expressions those are exactly the class of regular languages, so that gives the name regular expressions.

So, before I formally define what is a regular expression, we will try to understand what kind of strings are accepted by a regular expression, we look at some examples, and then we will look at the formal definition.

(Refer Slide Time: 01:14)



So, as I said regular expressions are an algebraic representation of languages. And in the real world, in practical cases, so they have any applications, so they have applications in your favorite - text editor, for example, if you are using a text editor like

word or any other text editor and you want to search for a word. So, the way the word application or the software goes about searching for a word in your document is basically by using regular expression.

If you want to search for a word, it tries to match whatever word that you are searching for in your entire document; and if it finds instances of that particular string that you are searching for, it will immediately tell you how many such instances are there and where they are present, so that is one application, in text editor to find strings.

Also another example is in compiler design. So, for example, when you write a C program, we put comments in our C program. And we know that comments are basically pieces of text, which are not necessary to compile the program; they are not directly useful in compiling and executing your program, but they are there basically to help the user understand what the programming is doing. So, how does the C compiler know which part of the text is a comment and how does it remove the comment from the mean program during compilation.

So, again there basically what is used is regular expressions. So, recall that in C program one way of putting comment is by giving the slash star string. So, if you write anything between a slash if front slash star, and star front slash that is treated as the comment. So, what your compiler does is before it compiles your program, it searches your entire program again it does sort of a string matching, and it tries to find occurrences of front slash star and star front slash.

And then whenever it finds such a pair, it just removes everything that is there between it. And does this for every such pairing of a front slash star and star front slash this is how comments are eradicated. So, these are two very common examples of regular expressions there are other places also in which regular expressions are used. But for this course, we will not be interested in the application side we will try to see what is the power of these expressions; in the sense that how are they I mean how do regular expressions define languages.

And what are these class of language, I mean we will prove that they are actually

exactly equal to the class regular languages which shows that anything that can be accepted by a DFA or an NFA does have a regular expression for it and vice versa.

So, before again I look at the formal definition, let us look at some examples of regular expression. So let us look at some regular expressions, and their corresponding languages. So, if I take the regular expression let say 0, so this is basically just the language consisting of the singleton string 0. Similarly, if we have the regular expression may be epsilon, so this is again the language consisting of the singleton string - the empty string or epsilon - the regular expression 0 union 0 1. So, union is symbols that are used in regular expressions. So, this consists of, so this is the language consisting of the strings 0 and 0 1.

Another way of writing regular expression is using the star operator, so 1 star corresponds to the language which is basically consist of all strings which have k occurrences of 1s, where k is greater than or equal to 0. So, if it is 0 occurrence that gives the string epsilon for a single occurrence we have 1 then 1 1 then 1 1 1 and so on. So, it is an infinite language.

And let us look at one more example, if we have a regular expression of the form let say 0 union 0 1 concatenated with 1 star, it basically consist of all strings that I can form by first taking either 0 or 0 1 and taking a string from 1 star and concatenating them. And I just write out all those strings. So, basically first I will have 0 concatenated with epsilon that will give me the string 0, then I will have 0 1 concatenated with epsilon that will give me the string 0 1, then I will have 0 concatenated with 1 that will give me the string. So, 0 concatenated with 1 is again 0 1.

So, I do not write it again then I will have 0 1 concatenated with 1 that will give me the string 0 1 1. Next I have 0 concatenated with 1 1 which again gives me the string 0 1 1 which I do not write, then I will have 0 1 concatenated with 1 1 which will give me 0 1 1 1 - three 1s and so on. So, these are the ways in which we can form languages out of regular expression. And observe that regular expression is just a finite expression; in the sense that it consist of a finite number of symbols, but it is capable of generating even infinite languages.

Now let us look at the formal definition of our regular expression. So, the way regular expression is defined is in an inductive manner. So, we will have some base cases, some base regular expressions, and then we will form regular expressions inductively using these base cases. So, R is said to be a regular expression, in short we write it as R E. If R has one of the following forms, so if it has one of the following seven forms that I am going to describe here. So, the first is it is so again I am going to describe it as I said inductively. Let me write it here; regular expression, the corresponding language and just let me keep this for comment.

So, if the regular expression is phi, so that corresponds to the empty language; the language which does not contain any string, so this is the first one. If the regular expression is epsilon, this corresponds to the language that contains only epsilon. So, an important point is that these two languages are different.

The first language is the empty set or the empty language, which does not contain any string; the second language is the language which contains the empty strings, so it is the language which contains a single string the empty string epsilon. The third is a regular expression ‘a’, where ‘a’ is any symbol in epsilon, so the corresponding language is the language which contains only one string that is the symbol ‘a’, for every symbol ‘a’ belonging to sigma. So, these are three base cases, now we will have the inductive cases.

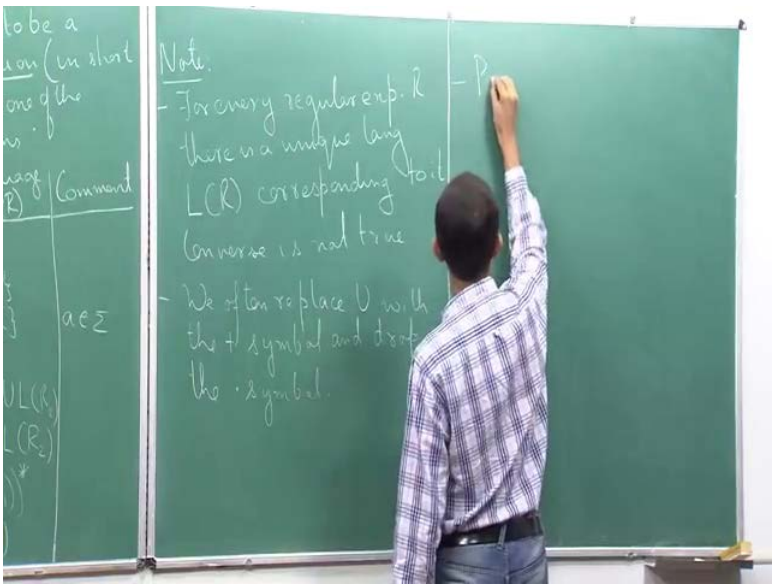
Now for two regular expressions R 1 and R 2, R 1 union R 2 is a regular expression; and the corresponding language is L of R 1 n union with L of R 2. We have R 1 dot R 2, which is L of R 1 concatenated with L of R 2. We have R 1 star, which is basically L of R 1 whole star. So, these are the basic, so I mean these are the main three inductive cases. And the last one is just to introduce bracket. So, we have R 1 with round brackets surrounded. So, this is nothing but same as the language L of R 1. So, what do we mean by L of R 1.

Basically this is the language of the regular expression R 1 and same for L of R 2. So, to be more precise, let me write it here. So, R is said to be a regular expression, if R has one of the following from, so this language that I wrote here. Let me just rewrite this. I

will just give a name to this. So, this is my L of R. So, whatever regular expression R that we consider, this is the language L of R, and this is how it is defined.

So, if we look at the example here for example, when we have 0 union 0 1 this consist of the language which has the string 0 together with union with the language which has the string 0 1, so 0 and 0 1. So, this is the formal definition.

(Refer Slide Time: 16:00)

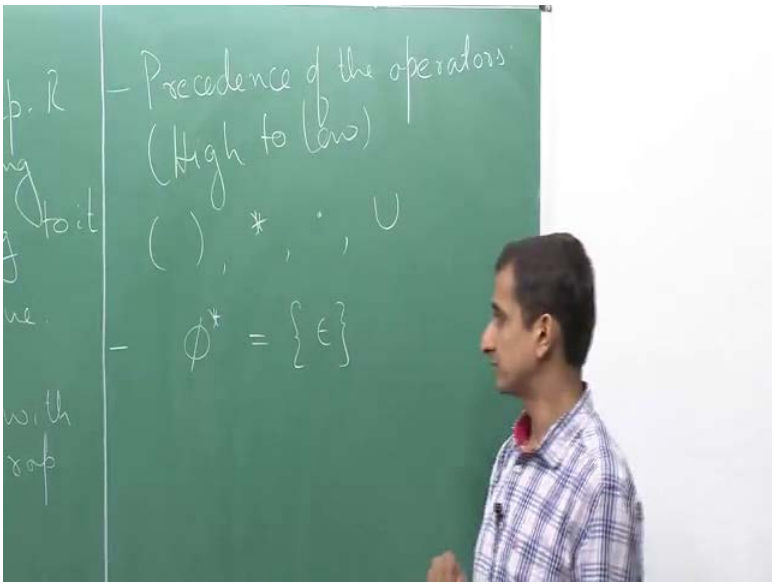


Now let us look at some properties and conventions of regular expressions that we often use. So, the first one is similar to what we said in the context of automaton that when we are given an automaton, there is a unique language that corresponds to that automaton, whether it is deterministic automaton or a nondeterministic. The same thing is true for regular expression as well. So, for every regular expression R, there is a unique language L of R corresponding to it, but the converse need not be true.

What do I mean by that that if you fix a language, you can have many regular expressions for that language. In fact, you can even have exponentially many or infinitely many regular expressions for a given language, it just depends how you decide to form your regular expression. So, often we replace union with the plus symbol and drop the dot symbol. So, just to give it more algebraic feeling, when we are

actually working with regular expressions, we replace the union symbol with the plus symbol. And wherever it is sufficient clear, we do not use the dot symbol we just ignore it.

(Refer Slide Time: 19:01)



The next point is very important; this is regarding the precedence of the symbol. Again like in algebra or arithmetic, when you are working with operators like plus, multiplication, division or other operators, and you form expressions using them, it is important that you have precedence law. So, you know that which operation you need to perform first, and which operation you need to perform after that, so it is between any two operators you need precedence. So, the same thing exists here as well. So, the precedence of the symbols or maybe I should say precedence of the operators from high to low is as follows.

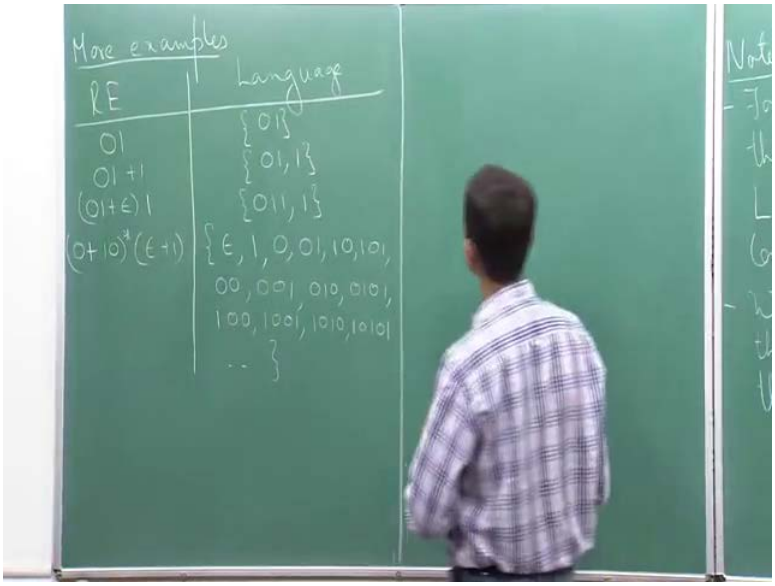
First we have the bracket operator. So, whenever we have something in brackets that needs to get executed first. Then we have the star operator; the star symbol. Then we have dot and finally, we have the union, so brackets, star, concatenation and then union.

And the last point that I want to make here is what is the language corresponding to the regular expression phi star. So, this is very important point. So, what is phi star? So, if

we look at the definition, so when we have star of a regular expression, so this basically is all strings that I can form by taking 0 or more strings from the language L of R 1. In this case, L of phi is phi, so if I take 0 or more strings from L of phi, so if I take 0 string that basically gives me so that gives me only one string which is that is the empty string, because the empty string is a string which has 0 symbols in it. So, that is the only string that can be formed by taking 0 star.

The language of 0 star is the language which contains epsilon. So, it is not the empty language, it is just the language which has the string epsilon. So, this very important and this you should always keep in mind.

(Refer Slide Time: 22:15)



Now let us look at more examples. So, I look at some more examples. So, we will have regular expressions on one side, and we will look at the corresponding language. So, the first one is let say the regular expression 0 1. So, we saw something similar earlier. This basically is the language which has only one string 0 1. Next let us take 0 1 plus 1, this has the strings 0 1 and the string 1. The third one is 0 1 plus epsilon concatenated with 1. So, how many strings are there in this language, and what are the strings.

So, again first I take one string from this regular expression, and one string from this

regular expression, and I concatenate them. If I take 0 1, and if I take 1 here, so the concatenation of 0 1 and 1 gives me 0 1 1; and the concatenation of epsilon and 1 gives me 1. This is the language that is then these are the only two strings that are there in the language.

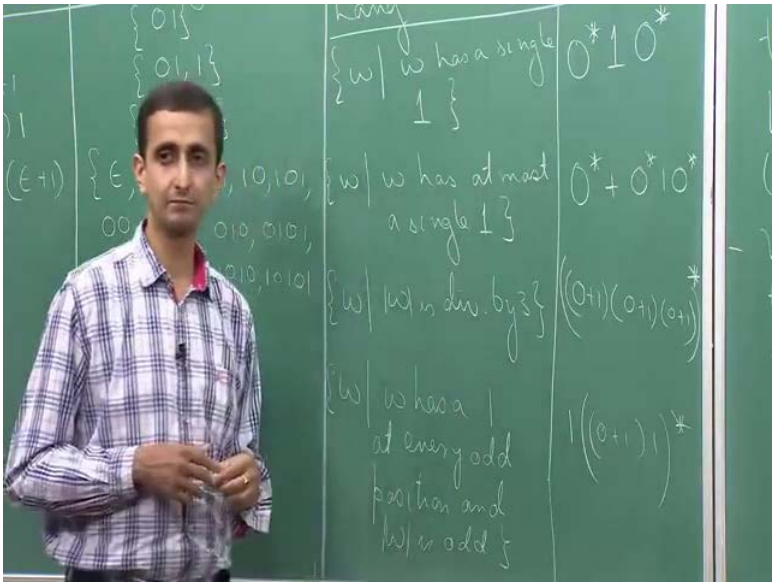
The last example is the following. Let say we have a regular expression 0 plus 1 0 whole star concatenated with epsilon plus 1. So, what is the regular expression in this case let us try to form. So, once again we will take one string from the regular expression on the left hand side and one from the regular expression on the right hand side and concatenate them. If I take so we have star here, so epsilon is a string. So, epsilon concatenated with epsilon gives me epsilon, epsilon concatenated with 1 will give me 1.

Next if I take strings, which have only one string from this set. So, if I take 0, 0 concatenated with epsilon gives me 0; 0 concatenated with 1 gives me 0 1. Similarly, if I take 1 0 from the left hand side, 1 0 concatenated with epsilon will give me just 1 0; and 1 0 concatenated with 1 will give me 1 0 1.

Now let us take two strings. So, how many ways can I form two strings from the left hand side. So, I can have 0 0; so 0 0 will give me 0 0 and 0 0 1. I can have 0 1 0 that will give me 0 1 0 and 0 1 0 1. I can have 1 0 0 that will give me 1 0 0, and 1 0 0 1. And the fourth is I can have 1 0 1 0 which will give me 1 0 1 0 and 1 0 1 0 1 and so on I mean now I will take three strings from this set, four strings and so on.

So, this is infinite language, but the basically the idea is that we whatever regular expression that is given to us, we try to break it down into parts that we can solve in an easier way or for which we can construct the language in an easier way. And then using whatever operators are there, whether it is a concatenation operators, or union operator, or star operator, we form the higher language that that is needed.

(Refer Slide Time: 27:26)



Now, let us look at some examples from the other side that is, if we are given a language, how can we form a regular expression out of it. Let say we are given a language and we want to construct a regular expression out of it. So, how can we do that? Let me define languages over here, and we will have the corresponding regular expressions on the right. Let say we have a language, so all our languages will be strings over 0 1.

So, will suppose we have a language consisting of strings w such that w has a single 1. What is the regular expression corresponding to this? So, if w has a single 1, the way to think the way to think how you will construct a regular expression out this is what will be the form of these strings, how will these strings look like? So, if w has a single 1, the string will basically be a sequence of 0s followed by the 1, and then followed by another sequence of 0s.

And the sequence of 0 in the beginning and the sequence of 0 in the end can either be an empty m, I mean 0 number of 0s or there can be anything that is higher than. So, the corresponding regular expression in this case will be 0 star. The first sequence of 0s followed by a single 1 followed by 0 star. So, this is the language or this is the regular expression for this language.

The next is suppose we have I just look at a generalized version of this language or a slightly general version. So, w has at most a single 1. So, all those strings which are at most a single 1, what does that mean that w can have either no 1s or it has the single 1. If it has no 1s it basically means it is just a string of 0s; or it has a single 1, in which case we know that this is the regular expression. So, because we have this or that is separating the two cases we write this as when w has no 1s, it is basically consists only of 0s; or it has a single 1, in which case the regular expression will be the following.

The third is when w is a string such that length of w is divisible by 3, so all strings whose length is divisible by 3. So, basically how do these strings look like. So, I will have a sequence of three symbols then I have another sequence of three symbols then I have another sequence of three symbols and so on, and these symbols can be anything. So, the way to write them is. So, the first sequence of three symbol can, so the first symbol can be anything between 0 or 1, the second symbol again can be anything between 0 or 1, and the third symbol is once again anything between 0 or 1 and this sequence keeps on repeating arbitrary number of times. So, I have a star around it. So, we have a sequence of three symbols followed by another sequence of three symbols and so on for as many numbers as we want.

Now let us look at slightly more complicated example. Let say all strings w such that w has a 1 at every odd position, and let us say length of w is odd. So, I want to look at all strings which have in this form. So, how do we go about? So, it has 1 at every odd position and the length is odd. So, basically the first symbol is necessarily a 1, because it is an odd position.

And then for every pair after that I can write the pair as the first symbol of that pair is a 0 or 1, and the second symbol is necessarily a 1, and this pattern keeps on repeating. So, I have either - a 0 or a 1, then I have a 1 and this keeps on repeating. So, this enforces that at every odd position, because this is odd, this is even and this is odd, I will have a 1 and the length is always odd. So, I will stop here today.

Thank you.