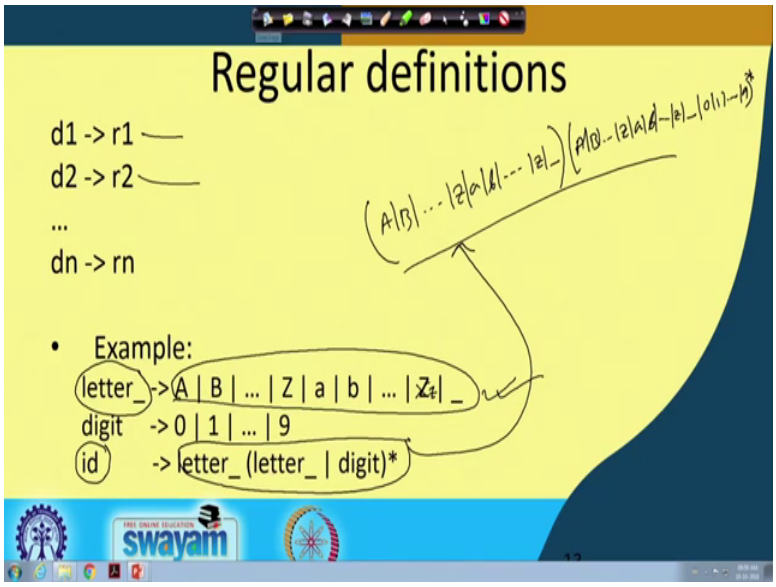
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 09
Lexical Analysis (Contd)

So, regular definitions so, we have seen few in the last class today we will continue with that at the beginning.

(Refer Slide Time: 00:23)



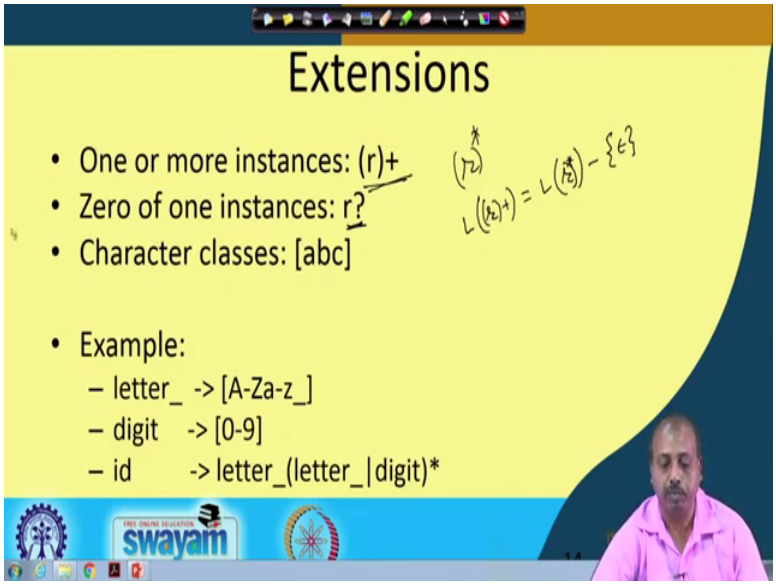
So, definition each definition for we have got a corresponding regular expression. For example, this d 1 has got a regular expression corresponding to each which is r 1 d 2 has got r 2. So, we generally write it in this format d 1 then 1 arrow and then the corresponding regular expression. A typical example is say this letter underscore is a definition and that definition is governed by this particular regular expression. And as we have seen in the definition of regular expression so, this is A or B or up to capital Z than small a small b up to this should be small z not big Z, this should be small z and the underscore character

Then the digit is another definition is 0 1 0 or 1 or 2 or something up to 9. Then id an identifiers another definition which is composed of the previous two definitions like letter underscore followed by letter underscore or digit whole star. So, the definition says of this id this definition says that any character string that matches this particular

definition should start with the letter or underscore character and that is if you substitute this right hand side of this part here. So, what you get is A or B or Z or underscore character followed by this characters or digits. So, this part this whole thing is actually something like this. So, this is A or B or Z or a or b up to small z. Then you have got the underscore character.

So, this whole thing followed by again this one that A or B or Z or small a or small b or small z underscore or 0 or 1 or up to 9 and this whole star. So, this is the full thing so, this is written in a short hand form in the here. So, for making the expression shorter so we normally followed this type of strategy that we follow we define some regular definition for some bigger parts. And then sub parts and then use those sub parts or sub definitions to define a more complex expression. So, this is the way we will proceed so we will look in to some more examples like shape.

(Refer Slide Time: 03:03)



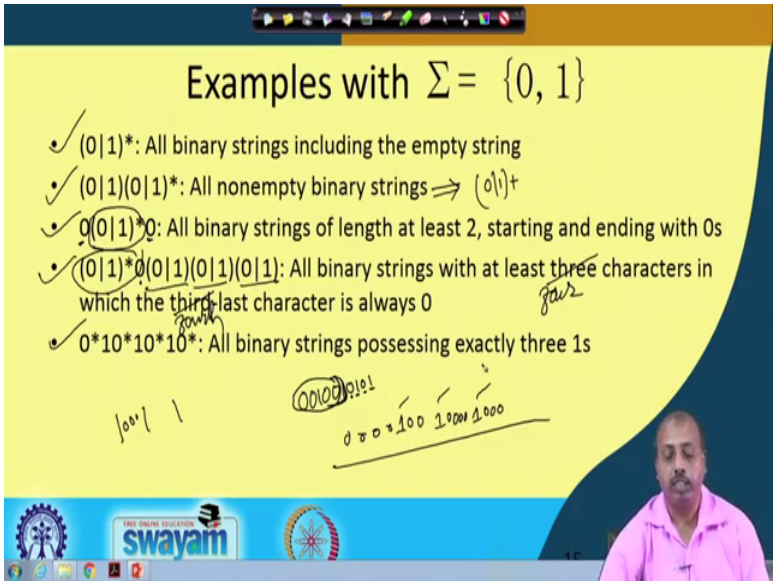
So, so, before that so, we have got some other notations as well like if I have a regular expression r, then we say that r star is 0 or more occurrence of r. Similarly, r plus is another notation. So, this one it says that one or more instances. So, it is the only the epsilon is excluded so, all others are there. So, basically language which is composed of r plus is equal to the language which is composed of r star so, language which is composed of r star minus the null string epsilon. So, then there r question mark so, this is 0 or 1

occurrence of r. So, this is it is not multiple occurrence so, it is 0 or 1 either 0 or 1 occurrence.

Then sometimes we define a character class for that we write like within square bracket we write abc. So, this means that I am talking about the characters a b and c and they are forming a class. So, this short hands will be used for writing complex expressions which otherwise becomes a bit ineligible to understand so, this short hands will be used. For example, this letter underscore previously you have seen a big definition. So, here it is shortened so, I am writing like A to Z so, this is a range A to Z then small a to small z. So, by character class definitions of this becomes one regular definition similarly digit instead of writing each and every digit so, it is we write as 0 to 9. Actually these are for human understanding so, as long as human being can understand the definitions so, it is fine. Of course, if you want to put it in a computer. So, or in some program then of course, you have to write explicitly a or b or c or d like that.

But for human understanding so, it is better if we use short hands so this is actually the usage of shorthand. Similarly, id is letter underscore followed by letter underscore or digit star. So, these are the some extensions that will be using while handling with handling the regular expressions.

(Refer Slide Time: 05:23)



Now, let us look into some examples, suppose I am considering strings over binary symbols 0 and 1. So, my in the language the alphabet set contains only two symbols 0

and 1. So, what is the language corresponding to the first regular expression? So, 0 or 1 star so, it means that if the star is not there so, it says 1 occurrence of 0 or 1 that is a single beat. Now, there is a star over this so; that means, that all binary strings including empty strings. So, everything will be acceptable in that language because, it says any occurrence of 0's and 1's with alternating 0's and 1's or the null string. So, everything is there so, this is the thing, the second regular expression. So, 0 or 1 followed by 0 or 1 start it is all non-empty binary strings. So, it so, this 0 or 1 star means 0 or more occurrences of the characters 0 and 1.

And before that there is another part of the regular expression which is 0 or 1 so; that means, the overall at least 1 0 or 1 1 must appear then rest of the characters they are may not be any more characters. So, single length string or maybe multiple length strings. So, you can understand that this actually equivalent to writing 0 or 1 plus so this is same as this expression. Then the next one say this one so, this expression says 0 followed by 0 or 1 star and followed by 0.

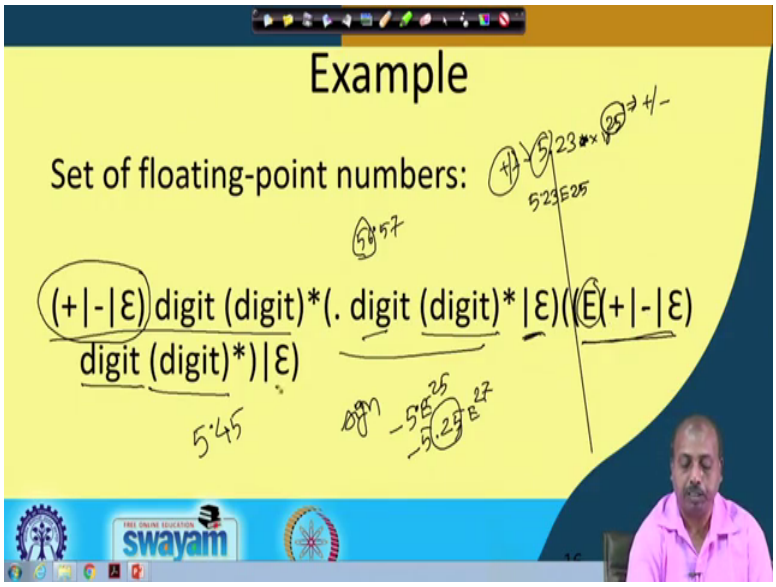
So, if we try to understand the meaning of this particular regular expression. So, this part means 0 or more occurrence of the characters 0 and 1, but the string must start with a 0 and end with a 0. So, the strings are having at least a length of 2 and it is starting with a 0 and ending with a 0. So, all binary strings of length at least 2 starting and ending with 0's are valid strings for this particular regular expression. Then the next one it is very tricky it says that 0 or 1 start 0 then the next three characters can be 0 or 1. So, all strings where the, which are having at least three characters in which the third last character is always 0. So, it is not three character so, it is at least four characters so, this is lightly wrong.

So, this is at least four character in which the fourth last character, fourth last character is always 0. So, I can have a string like 0 0 1 0 0 something like that 1 and then this is a 0 and then I can have three more character. So, it maybe 1 0 1 for example so, this 1 will match with this part this 0 will match with this part, then this 1 will match with this part this 0 matches with this 0. And then remaining part so this one this entire thing sorry up to this one, up to this one so this will match with the 1st part. So, this way I can have this particular regular expression which corresponds to all binary strings of at least four characters in which the fourth last character is always 0. Then the next regular expression it is 0 star 1 0 star 1 0 star 1 0 star. .

So, it says that there can be any number of 0's, but can be exactly three 1's in the string. So, for matching I must have 1 1 1 at the some places and in between I can have any number of 0's. So, I have three 1's 1 1 and 1 and in between I can have. So, may be at this phase I have got 2 0's at this place I have 4 0's after this I have got quite a few 0's so, like that.

So, any such string where I have got exactly three 1's will match with this particular regular expression. So, this way depending upon the language that we are talking about the word in it so, we can frame the corresponding regular expression. So, naturally so these are these examples are it are a bit synthetic. So, if we are looking for some programming language then we have to see like what are the valid words in that language and accordingly we have to define the regular expression.

(Refer Slide Time: 10:05)



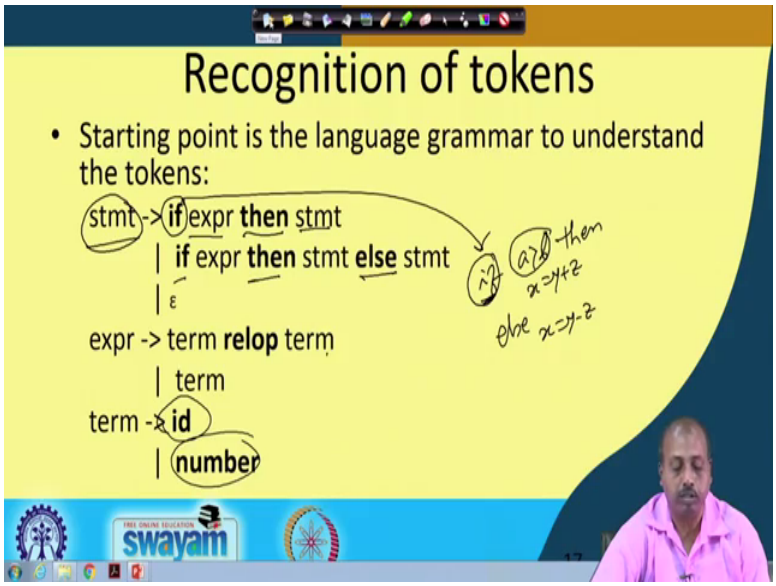
So, next we will look into another example say the set of floating point numbers. So, floating point number they have got several parts in it first there is a symbol. So, a number may be say a number may be say 5.23 into 10 to the power say 25 something like this. So, this is normally written as 5.23 E 25; now there can be signs also like at the beginning I can have a sign here that signs so, number may be positive or negative so, it may be plus or minus. Similarly this power so, it can also be plus or minus so, accordingly we define the regular expression the first part is defining, this first part is defining the first sign.

So, I start with the sign so, this sign part this plus or minus. So, is it is captured by this plus minus or epsilon. So, epsilon means there is no sign that is specified so it is taken as a plus number. Then the next part so, there is a decimal point and before decimal we have got some portion. So, here I have got a single digit so, that is digit, but it is not mandatory that there should be a single digit the number may be say 56 point something 57. So, this 56 will be captured by digit followed by digit star 2 digit and then this dot character is there. So, this dot appears and after that I can have the fractional part of the expression. So, this is digit followed by digit star or epsilon so, fractional part may or may not be there.

So, it can be that I may have say I may have minus 5 E 25 so, it may be there. So, at that point I do not have after 5 any digit. So, there is no fractional part or the number may be 5 minus 5.25 E 27. So, that way this 0.25 has to be captured. So, 0.25 will be captured by this part and here nothing is there so, that is captured by the epsilon part. So, this is there then the E character must be there the 10 to the power that exponent part. So, the E character must be there so that E is there then after that this exponent may have a signature may have a sign or may not have a sign. So, if there is a sign so, it may be plus or minus if there is no sign it is captured by this epsilon followed by again number. So, there must be at least 1 digit or there can be multiple digits. So, this whole thing can be there or this E part may be totally absent.

I may also like to represent a number like 5.45 without any exponentiation part this is a 10 power part. So, this part may be totally absent so, that is captured by this epsilon. So, this way this entire expression is taking care of the situation that set of floating point numbers in some programming language so, it may be captured by this particular regular expressions. So, you can formulate other regular expressions depending upon the words that are allowed in the language and accordingly come up with the recognizer for them.

(Refer Slide Time: 13:39)



Now, how do you recognize that tokens? So, starting point is the language grammar to understand the token. So, what are the tokens in the language so, that for that we need to look into the grammar of the language. For example, this is some hypothetical language where we have got this if then else statement.

So, this statement so, we will see later when we go to the parser chapter. So, this is this particular some words which are not some words in the programming language so, they will called non terminal symbols. So, this non terminal statement it is producing this terminals involve if followed by some expression then there terminal symbol then and then another statement.

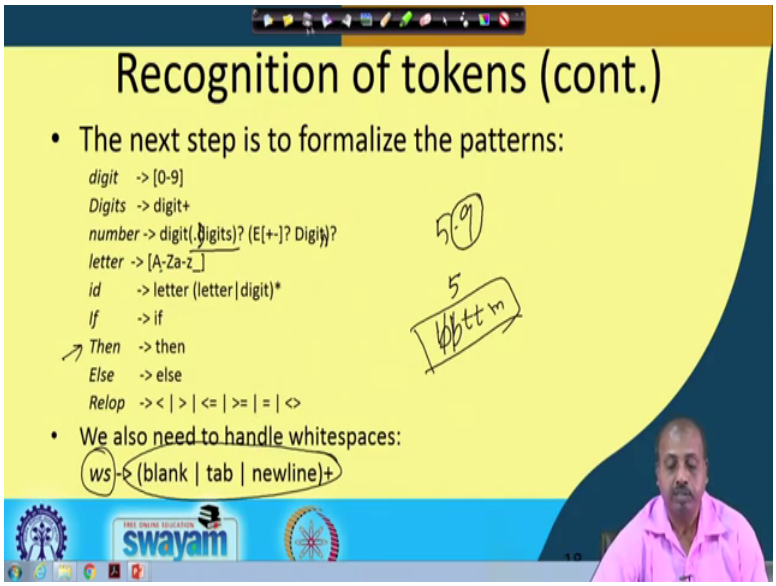
So, if expression then statement out of this if and then they are tokens of the language because they are some they correspond to some valid pattern that we have in our valid (Refer Time: 14:44) that we have in the program that we have that we take as input. Similarly, if in the second one if then and else they are the three patterns that are accordingly I should have corresponding tokens and or epsilon. So, statement may be if expression then statement or if expression then statement else statement or epsilon.

Similarly, an expression so I expect a conditional expression for the if then else statement. So, this is at some term followed by some relational operator and another term or it may be a simple term where term is actually an identifier or it is a number. So, I can have a statement like if a greater than b then some statement maybe x equal to y plus z

else x equal to y minus z. So, here this if will correspond to this token this (Refer Time: 15:45) in the input text file I have got these thing. So, this particular input stream portion so, they will give me the token if similarly this part will be reduced expression and how is it reduced.

So, you see that this is coming as term relational operators is this greater than and this b will be term. And this term will be represented by the identifier a and the second time will be represented by the identifier b. So, this will be more clear when you go to the parser chapter, but essentially what you want to emphasize at this point is to identify the tokens of the language. So, we have to start with the grammar of the language and then from there we can see like what are the important tokens that can come and accordingly we can proceed.

(Refer Slide Time: 16:35)



So, the next step is to formalize the patterns. So, what are the pattern like we have to give the definition, like digit is anything a range 0 to 9 the characteristic characters 0 to 9. So, they form a digit then we can put another definition digits which is digit plus that is 1 or more occurrence of digit.

Then number is defined as the digit it should start it should it must have at least 1 digit and then dot digits, digits and question mark. So, digits and question marks so, this part we will mean that 0 or more occurrence of digits so, it may be 5.9. So, in that case this point dot digits so, they will match with this part and sometimes I may write simply as 5.

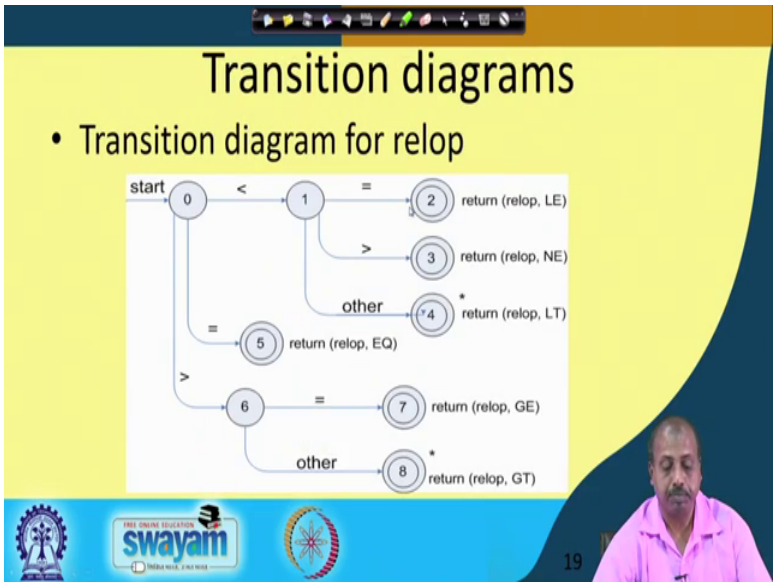
So, there is a integer number so, there is no fractional part. So, this question mark will mean 0 or 1 occurrence of digit. And after that I can have I will have E the exponentiation then there may be a question mark and this plus minus. So, occurrence of plus symbol or minus symbol and then digits so, this would be digits so, I can have more digits there.

So, this way I can actually this should be capital D capital digits ok. So, this should be so, this way I can we have already explain the floating point number so, on the same line this number maybe define. Similarly the letter so, we have got the A to Z capital uppercase letters A to Z then lowercase letters a to z and the underscore character. Then an identifier when you are defining so, it is letter followed by letter or digit star. Then this if is another definition which is actually the characters i followed by character f appearing on the input stream. Then I am defining another definition then so, which is the characters string t h e n this character. So, this way I can define this thing then say relational operator. So, I can have less then greater than less or equal greater or equal, equal not equal like that so, we can define the relational operators.

We also need to handle whitespaces. So, whitespace how do you define a whitespace? So, blank tab and new line character. So, these are the three whitespace characters we have now there can be 1 or more occurrence of such whitespaces. Somebody may write like say blank then may be put in 2 tabs and then a newline character so, that can happen.

So, this whole thing will be identified as a single whitespace and may be the lexicon analysis tool will remove all this whitespaces from program. So, that can be done. So, this whitespace so, this whitespace is another regular definition which is given by this regular expression blank or tab or new line plus so any number of them. So, this way we can define the patterns and the corresponding tokens.

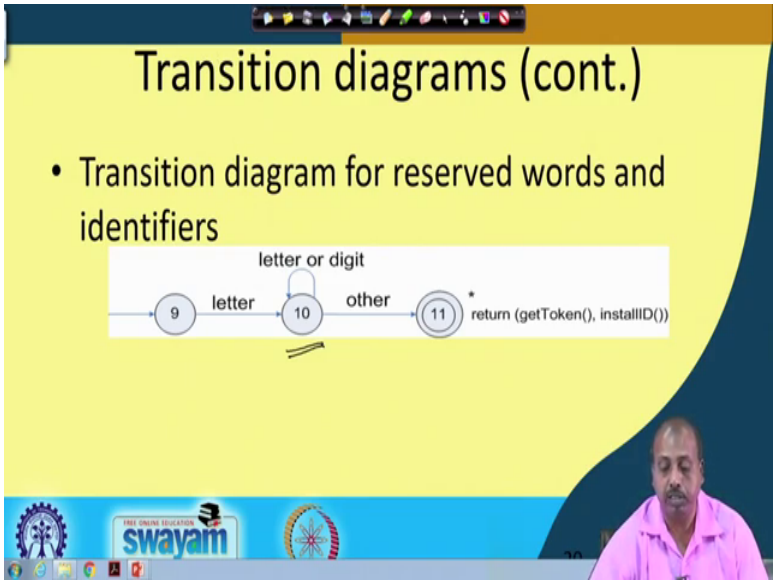
(Refer Slide Time: 19:55)



Now, once we have done that so, we can you take help of some transition diagrams to identify the tokens. So, for example, if I am looking for the relational operator the definition was like this less than greater than less or equal greater or equal, equal and not equal. So, accordingly I can say that if this is my if the lexical analysis tool it starts at state 0 of these finite automata. Then or getting the less than symbol it will come to state 1 and then if it after that in the state 1 if it sees as next input symbolize equality then it comes to a state 2 which is a final state and identified by 2 concentric cycles and then it will return the token relop with the value or the attribute as LE. So, LE maybe some less or equal so, there may be some value or some constant define for the symbol for this symbol LE.

But, whatever it is so, conceptually we can say that the token delay relop. So, it can have values like LE NE LT etcetera so, it is returning the value LE. Similarly, after coming to state 1 if it is finds is greater than symbol then it can say that it is relational operator not equal to. So, if it is any other symbol so, it is less then that so, it will return the relational operator LT. Similarly, at state 0 if it finds equality so, it will return relational operator EQ and then it is greater than equal to then it will return the relational operator greater there was it will return the relational operator greater or equal and here in state 8 it will return relational operator token with value greater.

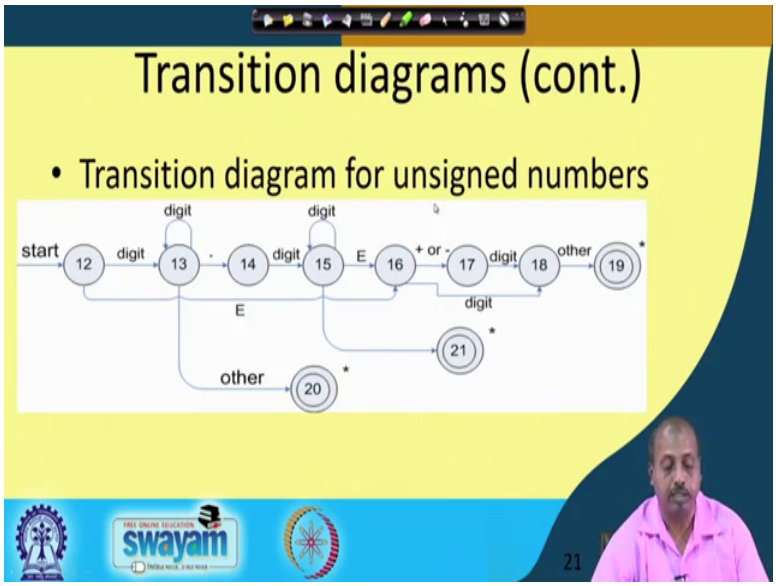
(Refer Slide Time: 21:59)



So, in this way you can define other such transition diagram also like this is the transition diagram corresponding to the reserved words and or identifiers. So, this is the starting state then it is coming to this letter followed by letter or digit. So, here and then if it is not matching with if it is not matching here then it will anything else. So, it will go to this other states. So, if it as long as it gets letter or digit fitted with looping there when it gets some other symbol it comes to the state eleven and it returns get token and install id.

So, this get token will get the corresponding token. So, depending upon this ID so, this will be returning the ID and this install ID. So, it will install the identifier into the symbol table and that symbol table off set will be returned to the partial. So, this way I can say that this transition diagram for reserved words and identifiers can be developed. So, we can use this in the lexicon analysis phase for defining these keywords.

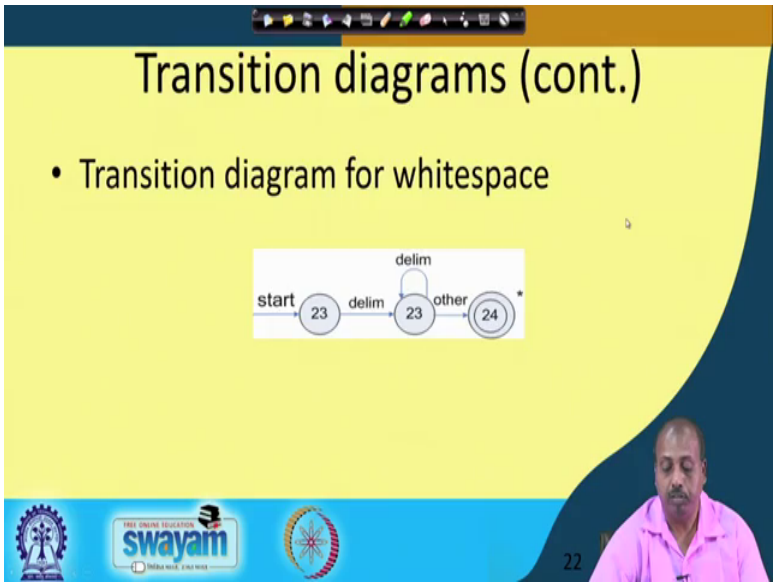
(Refer Slide Time: 23:09)



Then we can have unsigned numbers. So, this is another possibility like from the start if you if you come to state 12 and if you get a digit there so, it comes to. So, it is digit then digit start then dot so, this number definition that we have seen previously so, it is for that purpose. Now, it may get only a single digit and that may be the end. So, it comes to state 20 it may be that from the beginning. So, you do not have any digits immediately starts with 10 power that is E. So, a straightway come to here come here. So, this way we can represent the valid unsigned numbers in the language in terms of a transition diagram. So, regular expression is a definition for the tokens that we have in the language.

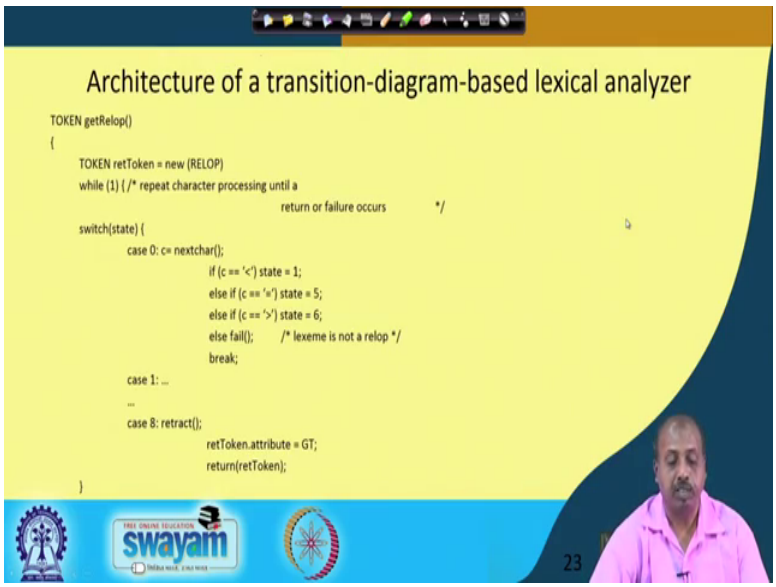
Then their implementation can be in terms of transition diagram and once you have got a transition diagram. So, you can realize it by means of some program or you can realize it by means of some automata. So, this is some sort of automata because I have got a number of states and transitions between them. So, that defines 1 automata so, we can us that symbol that technique.

(Refer Slide Time: 24:29)



Next for white space so, it is delimiter followed by any number of delimiters and any other symbol it will come to state 24 where it will return the token delimiter so, this or white space. So, this is this is what this delimiter is this whitespace transition diagram may be developed like this. We can so, in this way so, these are a few examples only so, for depending on the programming language that we are handling.

(Refer Slide Time: 25:05)



So, we have to see: what are the different regular definitions and accordingly we have to do it. Now, architecture of a transition diagram base lexical analyzer tool so, this is just a

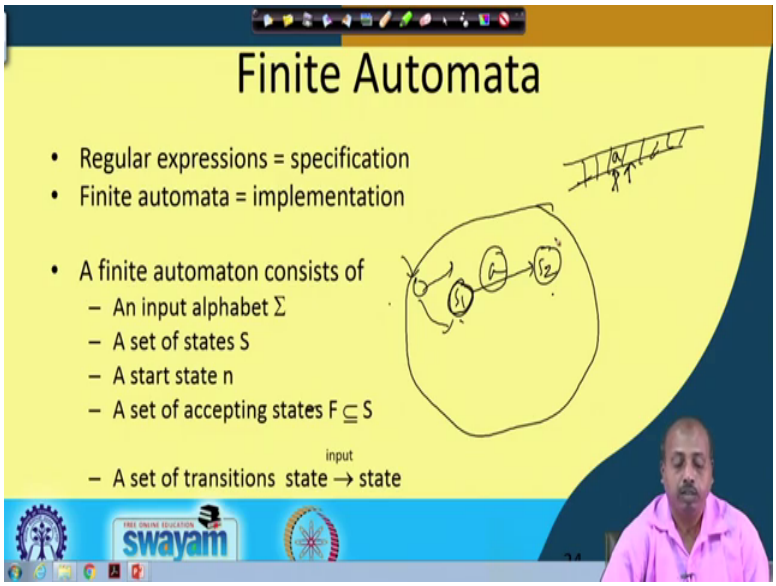
just a typical example ok. So, how we will how can we develop a corresponding code. So, this token get relop so, this is the relational operator for the relational operator part so, it will do it like this. So, it will first it will return a new relational operator as the token. So, while so, now, it will read the characters until a return or failure occurs.

And depending upon the state so, it is case 0. So, this relation actually we need to consult this relational operator table, a relational operator transition diagram so this one. So, you see that I so, it is depending upon the states it is 1 or if it is 1 means I have seen this less than symbol and then we are writing it like this sorry case 0 case, 0 means we have not seen anything now that is in state 0 so, that is in state 0 ok.

So, we are not seen anything so, we are correctly in state 0. So, the possibilities I will see a less than symbol, equality symbol or greater than symbol. Accordingly you can write this code the case 0 so, get the next character if the next character is also less than then it come it comes to state 1. And if it is equal to then it goes to state 5 it is greater than goes to state 6 otherwise it is a failure. So, lexeme is not a relational operator; similarly at state 1 also you can write a few, write a few rules here. So, like here so, state 1 so, I can see the equality greater than other symbols so, like that.

So, similarly this sometimes will be requiring to come back. So, state 8 it will be doing a retract because state 8 is here. So, you are getting something else so, you need to come back so it is not there not a relational operator so you need to come back so, here so, this is attribute is GT. And so, it will return the token relational operator token; this is a based on the transition diagram you can write a piece of code which will correspond to realization of the transition diagram.

(Refer Slide Time: 27:35)



Next we will look into something called finite automata because transition diagram is a it is a way of doing the thing, but it may not be very much suitable very much powerful in terms of the representation. So, we would like to represent them by means of finite automata. So, regular expression is the specification for the token and finite automata is an implementation of the token.

So, a finite automata it has got an input alphabet set sigma a set of states S a start state n a set of accepting states F which is a subset of S and a set of transitions state to state on some input value. So, it will if it is currently at state S 1 so, if you have a transition from state S 1 to state S 2 and it is on some input symbol a then it can go from state S 1 to state S 2 by consuming this input symbol a. So, what happens is that this automata is specified by means of a diagram like this where starting with the start state I have got all this transitions and there is an input stream ok. So, at present maybe the input pointer is somewhere here.

So, if you are at the state S 1 and the current input symbol is a then the input pointer is advanced to the next position and the current state of the system becomes S 2. So, it consumes the next input and proceeds to the next states it transits to the next possible states. So, this is the idea of finite automata. So, you can use these finite automata to realize this lexical analysis tool for designing the lexical analyzer.