

Indian institute of technology
Kanpur
NP-Tel
National Programme
On
Technology enhanced learning
Course title
Compiler design
Lecture-05

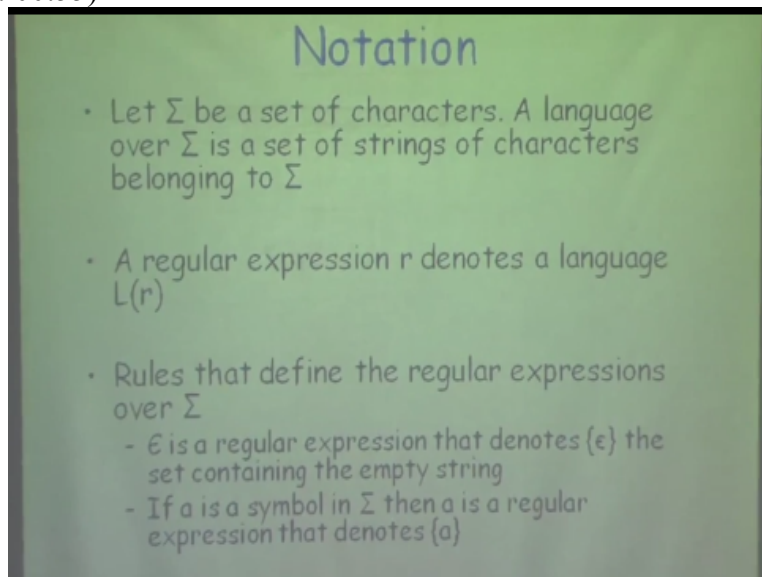
by...

Prof. S.A, Aggrawal

Dept. of computer science and engineering.

So let us talk about discussion from where left yesterday and we started looking at how to the design lexical analyzer want suppose to do and what are the kind of issues we may face by designing it and one issue we looked at also t format for this is format for talking looking at the keywords for lock and we were looking at some examples of TL 1with found a keywords for deserved and what kind of problem it can give guys there are some examples where you find that reading them pasting them and understanding them will where some skills of straight form okay another issue in the similar language maybe take something like you would like to declare.

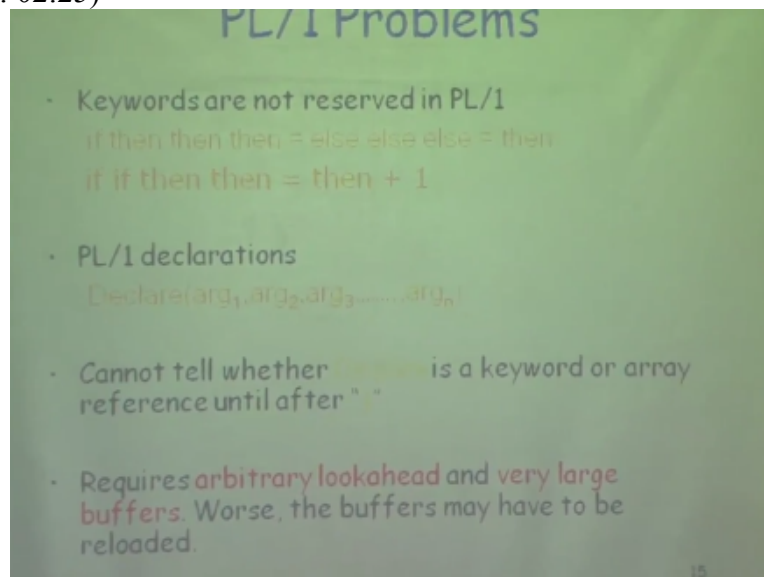
(Refer Slide Time: 00:55)



Declare with a keyword which said back want to declare some variables of certain time so I want to give a list of arguments here and N and I must say: in the end I may say that all these are also possible that actually this is a function passing all these as parameters okay when why figure out that dealing with keywords which is declare what I am dealing with nary reference or I am dealing with to see our function syntax and theory will be safe okay

So that means I will have to have some arbitrary look ahead because this list of argument can be arbitrarily long and I have to have some arbitrary look ahead where I say that when I counter character of the right : then only I know the weather I am dealing with a declaration and dealing with array and I dealing with up to see here and so on okay now this problem becomes really difficult because if I am saying that remember that to saying the I am going to design a lexical analyzer and which is going to be a buffer and my input point of we were just moving right over the buffer make sure that I am reading something and I am getting something which means putting back to the input stream by this arbitrary lane by it is possible.

(Refer Slide Time: 02:25)



That the whole purpose and keep moving and then after flushing many buffers I uncounted this buffer character and that point of my decide that I already flush this buffers that means but first that means because of arbitrary look ahead either I need ready not go first and even those buffers may not be sufficient so I may have to remove the buffers so I gave you some examples of 410 and TL 1 now people may argue.

Let some Greek languages nobody uses them and therefore this problem does not passed fully so question is have we resolved this issue on you know some examples of languages which are very prevalent today which are used commonly and still have these kind of issues where I cannot just find out without the context or kind of so lexis man being with anyone having example in money so this is an example from C++ okay we have these syntax for template okay.

(Refer Slide Time: 03:36)

Problem continues even today!!

- C++ template syntax: `Foo<Bar>`
- C++ stream syntax: `cin >> var;`
- Nested templates: `Foo<Bar<Bazz>>`

And then you have put it outputs and now but I have less pick templates what is it is it coming the: nesting or because of output I do not know deposit so it is not that you know resolved all these problems only thing we have to remember is that all these problems are described so far they cannot be resolved just by using lexical language I need to know something about the tokens which accrue in the context which means I have to shift this problem.

This part of the problem the syntax analyzer before I start finding out that what kind of symbols I am B unit so if the symbol occurs in this context then I know see when I am just tokenizing it I will not know because I have already tokenized have know the information and tokenized this and there no clue of what we tokens are okay if it is in this context then I will say that it is coming with a nesting that actually deserve who write the rules.

This is coming the fallow with this syntax and this is output but context I will know only even I and use sub sequenced and so most of these problems cannot be handle by lexical analyzer alone and need to do something more that means passing this information to syntax analyzer and then what the tokens will be is this issue cleared with everyone okay so question that comes is now so let's now get into implementation of what you want to analyze here or how we want to tokenized and how do we describe tokens.

(Refer Slide Time: 05:20)

How to specify tokens?

- How to describe tokens

2.e0 20.e-01 2.000

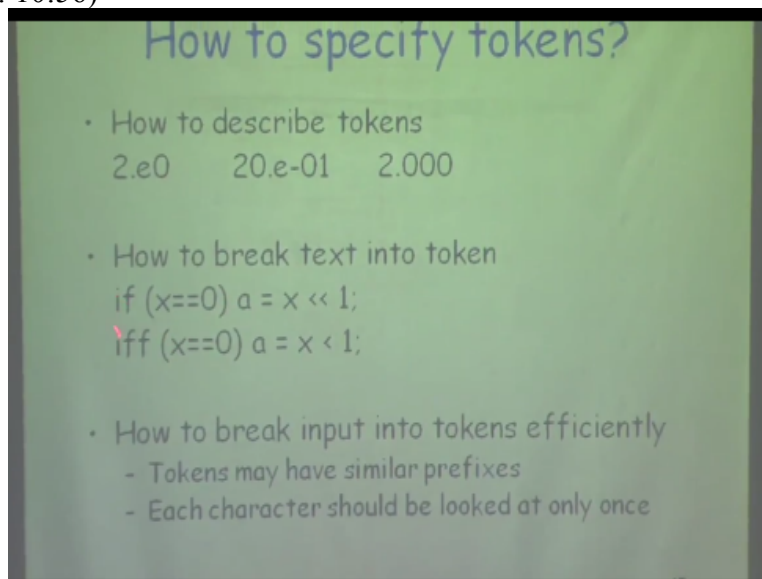
So here is some set of Lexis is which are basically numbers so I have a floating-point number here another number and yet another number and we want to now bake this text so this is the first problem and it we are going to face him you want to break this into a sequence of tokens and there is yet another example now one of the difference between so if I look at this what does it say so this is saying if X is equal to 0 then a the sign X and then what happens here because of the priority of the operators of dissidence of the operator I will say that I want to first do a left shift on X and then I assign it to X and here we are just saying this Boolean but here if you see. There is an additional now how do I break this is it if and then followed by a variable or this iff is the variable me on this because you were dealing with languages I give you an example yesterday.

(Refer Slide Time: 07:46)

$D0|0|5 = 1,25$
Specification
Implementation

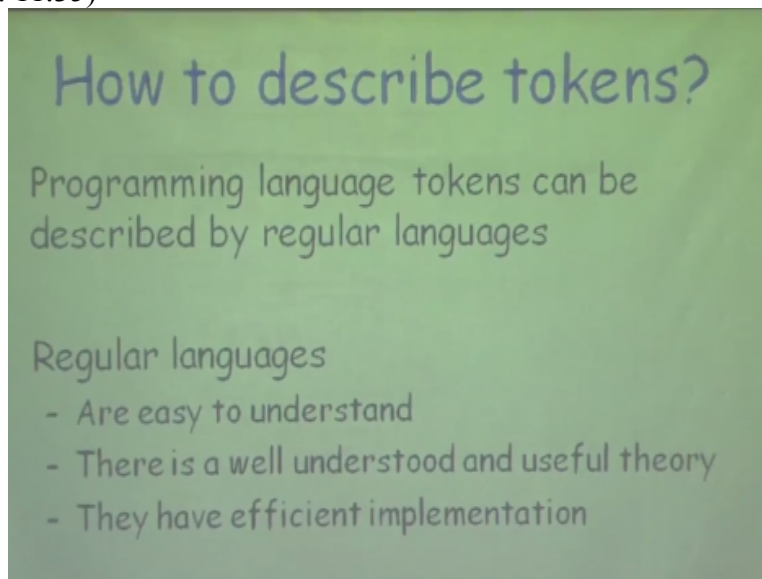
Then I have considered this as saying that this is an identifier the compiler would have required two parts token us languages is going to play a role in somewhere the implementation issues is coming so what are the two parts one part is the specification and other part is implementation I mean that is possible we were discussing previous points but suppose lexical analyzer can make a decision okay then there are some issues which are going to be implementation issues. So let me give you another example of implementation issues when I give specifications of tokens like identifiers characters followed by a number right now if I say X 10 is an identifier and X 100 is an identifier okay now should I start saying that since I already know X 10 is an identifier okay maybe it's already in the symbol table this also remember that this also matches specification X could be an identifier. So can I tokenize it there and say that is actually an identifier followed by a number we should not write but that is a matter of implementation because this matches specifications but somebody can say well X is also identified a fact just saying that X make x and then say 100 and then subsequent phases will say oh that is an error that is not possible So let us keep on discussing these issues how to break input tokens and efficiently you say that example and pushing.

(Refer Slide Time: 10:56)



One is that the tokens have may have similar defuses like here like given an example so prefix is singular when we have this and we actually want to look at each character only once so this idea of the keep on pushing keep on moving my point over the inter that is going to be a over that is going into a buffer are to handle it very efficient continuing on our description now tokens can be

described by that word language and your regular languages are and obviously they have certain properties they have any also really handling the regular languages.
(Refer Slide Time: 11:35)



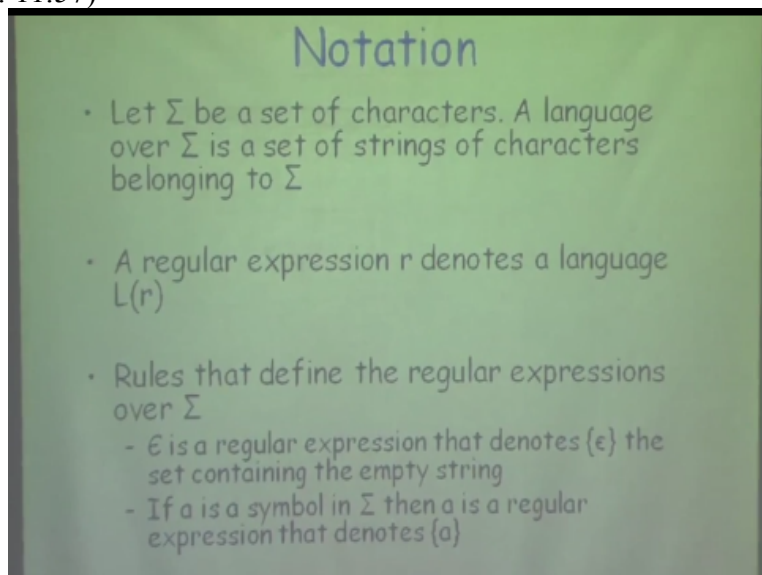
How to describe tokens?

Programming language tokens can be described by regular languages

Regular languages

- Are easy to understand
- There is a well understood and useful theory
- They have efficient implementation

And the familiar with regular languages are and obviously that have certain properties that have any nice mathematical theory which are develop on this and we can develop also most of really and handling the languages and this since we have discussed this with the large retail properties of regular expressions I just assume that notation we are going back to the notes on the books.
(Refer Slide Time: 11:57)



Notation

- Let Σ be a set of characters. A language over Σ is a set of strings of characters belonging to Σ
- A regular expression r denotes a language $L(r)$
- Rules that define the regular expressions over Σ
 - ϵ is a regular expression that denotes $\{\epsilon\}$ the set containing the empty string
 - If a is a symbol in Σ then a is a regular expression that denotes $\{a\}$

And notation we are going to do this is very similar that string and a sigma is the set of characters and sigma language and then a regular expression R denotes actually a language a lot and then we say that I can now define rules using the standard operators rules I have in my

regular definitions so simultaneously is a regular expression which we say that is the non language.

(Refer Slide Time: 12:32)

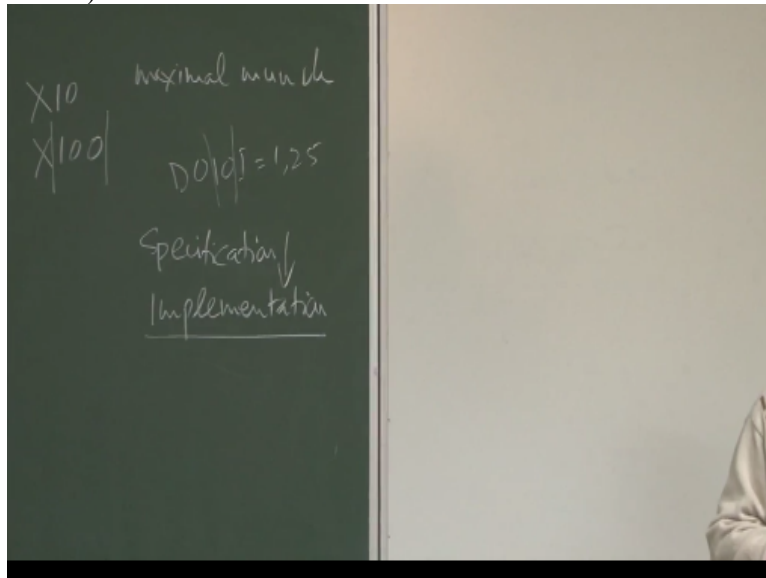
- If r and s are regular expressions denoting the languages $L(r)$ and $L(s)$ then
- $(r)|(s)$ is a regular expression denoting $L(r) \cup L(s)$
- $(r)(s)$ is a regular expression denoting $L(r)L(s)$
- $(r)^*$ is a regular expression denoting $(L(r))^*$
- (r) is a regular expression denoting $L(r)$

And A is a symbol denotes A is a symbol Σ then we say that this expression that could be denote the regular language and then use operators so if we say that r in Σ^* does r^2 regular expressions and then if we say that LR and LS and is specified by R and r in this specified out. And if it define it non technician operator let I am say that r followed by s is actually union of languages LR and LS and if I take so this is R operator this is non technician and then I can talk about Boolean operations and we said this is just for sleek of revision these are the operators and we will be use for specifies regular definition or regular language okay.

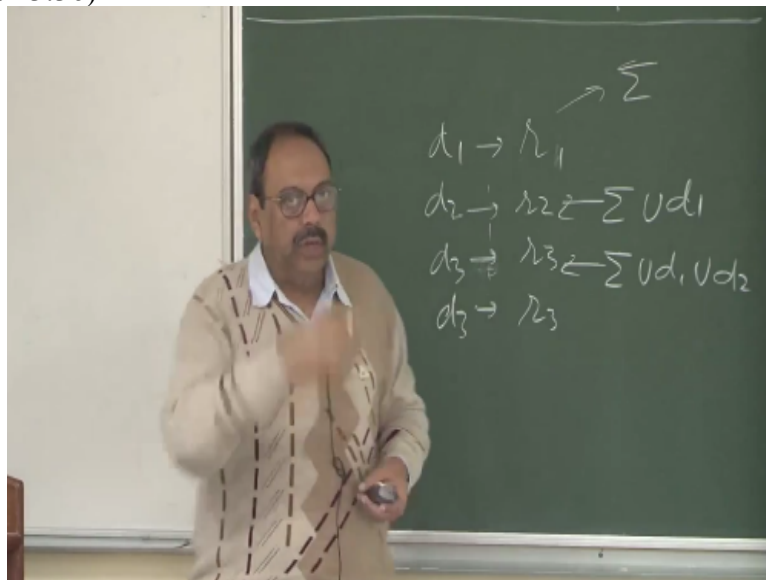
(Refer Slide Time: 13:15)

- ### How to specify tokens
- Regular definitions
 - Let r_i be a regular expression and d_i be a distinct name

And there are examples take so what we want to do this we want to use something known as regular definition this is something we can understand because now .
 (Refer Slide Time: 13:29)



We are getting into an issue of implementation and we have moving away from and we are moving from array it off specification so what we want to say if I have regular expression R.
 (Refer Slide Time: 13:36)



Some regular expression I want to give a descriptive name because I do not want to keep on using these regular expressions in expansion I want to give a descriptive name at this is I can see almost like a similarity using to write something like a function you see the truth there is some of needs to be repeated again and again and therefore I do not want to do that work of repetition I just give a name and can we do a possible are there that is the closed similarity I can see here that every language expression I am going to give a name rather than using R_i over and over again okay.

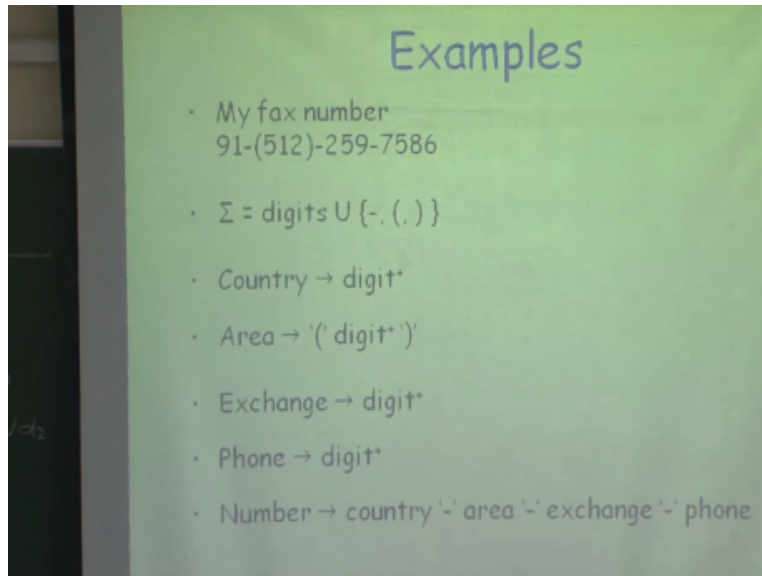
And then we will say that if I have a regular expression let see r_1 I will give name to this d_1 and then I have a sequence of these that more expressions and let me give these names and the only property I need to worry about here is that whatever is my first definition in this sequence here that is the regular expression over the sigma yes this is regular expressions which is sigma but can I look at R_2 because I already have d_1 here I given a name this regular expression I can say that this is actually a regular expression work sigma will be work okay.

It is a symbol in a it is a symbol in Sigma then we say that this expression and then I can use so if we say that the language and so this is this is and then I can talk about so this is I can see that every regular expression I am going to give a name rather than using our I over and over again okay and then we will say that if I have a regular expression let us say r_1 I give a name given to this and then I have a sequence of these regular expressions and let me give these names that whatever is my first definition in this sequence that is a regular expression over Sigma okay.

So this is a regular expression which is over Sigma and I you get R because I already have a d_1 here so I have given a name with this regular expression I can say that this is actually a regular expression of workers Sigma n b_1 and if I look at r_3 because these names are available to me I do not want to repeat so this is like d_1 and d_2 can you can be replaced by these regular expressions I do not want to repeat it here so all I am saying here is that this is regular expression over Sigma D_1 and D_2 and so on okay.

So I can have the sequence of regular definitions so what we say here is that regular definition is a sequence of this form okay we are the only property follow Is that if I look at any regular expression R I this is a regular expression over Σ and all the definitions which have already accrue so I do not loose definitions and I do not lose any context because we are saying is that I just do not want to repeat this regular expression again and I use a very concise notation that is the only thing is happen right okay.

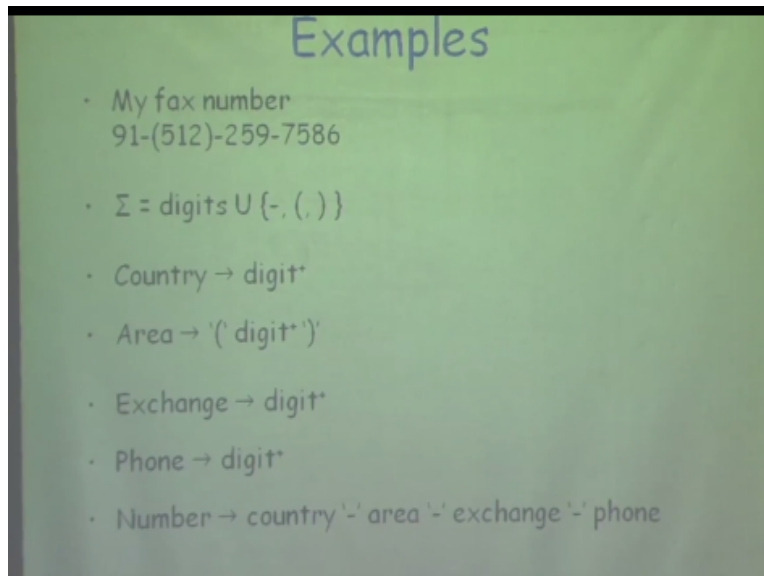
(Refer Slide Time: 16:03)



So here is a fax number now if I want to describe this so I will give you an example that I suppose I have a numbers and I want to find out all the quainter information here okay I can describe this by saying that what is my Σ consists of digits and our special characters left and right back again and then I can say that actually consists of 24 which is a string of digits and then I have area port which again is a string of digits.

But which I put and then I have exchange information which is a list of digits and then I have the phone number which is seven five eight six and then I can describe I can say that every number must have this form that is really the very point in fact I mean one you can here you see here is that when you actually make a call from where we just send a string of numbers and when they do billing they are being able to they are able to find out to which number you made a call you can find out the better this is ISD call or STD codes I can just process that then you can write the program and you can process all that information but then this is an implementation issue that comes okay.

(Refer Slide Time: 17:21)



What is the valid country code for example I am talking about now all this information is a regular expression but I will say for sake of implementation that no country code for any more than of similarly if I want to put certain restrictions or no area code any more than of support addition so then the experimentation issue comes in and tools we have they will provide you certain more ways of specifying and say that this can be a digits a string of digits of know the spring two or this could be a length of digits of at most be phone okay.

So I can also put these kind of additional specifications this is not part of regular expressions but this is part of the tool are going to use for regular implementation you remember that in like slide tool you could put this kind of restriction saying that what is the maximum length of string I can have between right we have got somewhere beside that an identifier can we of maximum length 32 right now for implementation I need to remember that and then this can be part of your specification is it safe that if your country code.

Is say more than two digits long and sigma so this is where slowly we start getting into the implementation issues saying that we have some specification spark for sake of experimentation we need to have more precise information about what will be valid token and all okay so I am not going to have an arbitrary long string of characters and say this is an identify I am going to put a restriction that whether it is way 16 or 32 so this issue clear to everyone okay so more examples then before I get into programming languages.

(Refer Slide Time: 19:12)

Examples ...

- My email address
ska@iitk.ac.in
- $\Sigma = \text{letter} \cup \{ @, . \}$
- Letter $\rightarrow a | b | \dots | z | A | B | \dots | Z$
- Name $\rightarrow \text{letter}^*$
- Address $\rightarrow \text{name '@' name '.' name '.' name}$

So here is my email id and if I want to describe this as using regular definitions okay what is my character set here it is a letter and then I have dot and symbol set this is my character set and then I can say what are my letters so I can rather than using these alphabets over and over again I can give a name to this like all this is better and then I say that name is looking for the string of letters and address is nothing but a name forward by this symbol forward by name and so okay. Have you look at you can have different descriptions you must say that this is a domain information's and so on okay and you can write different kind of descriptions of this and the they one description follow the restrictions not only that okay.
(Refer Slide Time: 19:58)

Examples ...

- Identifier
letter $\rightarrow a | b | \dots | z | A | B | \dots | Z$
digit $\rightarrow 0 | 1 | \dots | 9$
identifier $\rightarrow \text{letter}(\text{letter}|\text{digit})^*$
- Unsigned number in Pascal
digit $\rightarrow 0 | 1 | \dots | 9$
digits $\rightarrow \text{digit}^*$
fraction $\rightarrow \text{'.' digits} | \epsilon$
exponent $\rightarrow (E ('+' | '-' | \epsilon) \text{digits}) | \epsilon$

So let us get into more examples and now I want to get into something which is closer to what have going to implement of part of a calls and programming in that so I want to give

specification for an identifier so identifier is something you have been discussing right from the beginning so identifier is nothing but it comes this off let every minutes and if obvious parts with a letter followed by at least one letter or digit.

It is consist of at least and you followed by zero or more like and this and so this is the kind of specification I can have four identifiers so what you see here are combination of specifications and combination of their definition itself so I have just taken this and I have given a name to this later this is something which we have used you could have used to be valuable here okay.

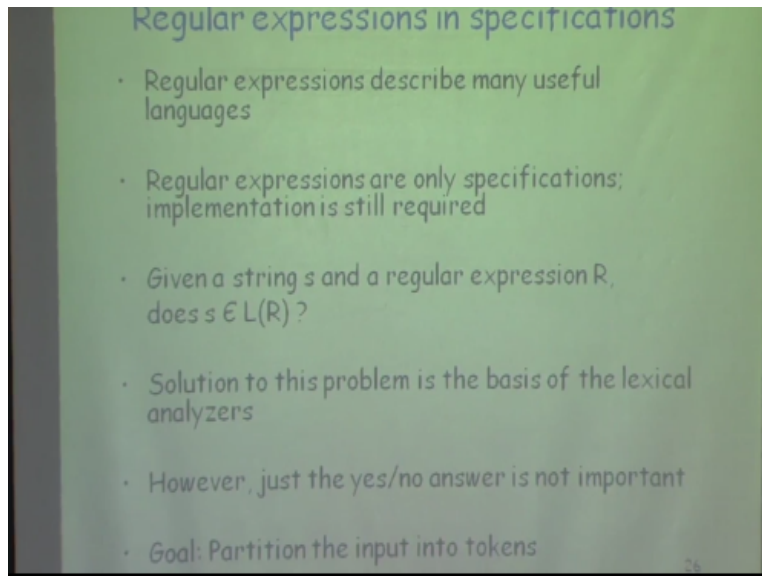
Which is distinct we do not here about it later something which symbolically immediately let me I can start reading my specific over that I am going to be but know nothing I could have been anything there okay so here is a letter specification which says letter is nothing but it consists of either A or B so all lowercase in all uppercase letters a B okay and similarly all the digits from zero to nine and nine to five can be specified standards okay.

Similarly if I want to write say unsigned numbers in Pascal then I am saying that I have these so these are floating point numbers of your numbers I have these digits and so digit is 0 to 9 and then I define another name which I say is consists of digits at because least modification we can all use least modification so can long use least modification we can long use 0to 9 I used on new operator plus which we said that consist of the at least one.

Digits can be empty and then fraction part is going to consist of dot followed by at least one digits of fraction maybe completely missing in my number and then I may have exponent and exponent I will have this letter E followed by an optional science and we are saying that the sign of the exponent could either be plus or minus or it could be missing and if it is missing and these going to be a reward of reputation and we can that is positive and then if I am using this notation then I must have at least one digit.

So I cannot have numbers like this is invalid so if I have E I must have at least one day you could say that one is exponent value or the whole exponent may be missing so that optional I am putting in epsilon here and then if I want to say number the number is consisting of nothing but they get followed by a fraction followed by exponent so this phase I can keep on developing the complete set of regular definition for whatever language I have this discusses is this clear to everyone when you have already done this when you were losing lexis and you already know that how to special use regular expressions assuming that this is not something which is very new it's just a matter of rotation will we move ahead.

(Refer Slide Time: 23:25)

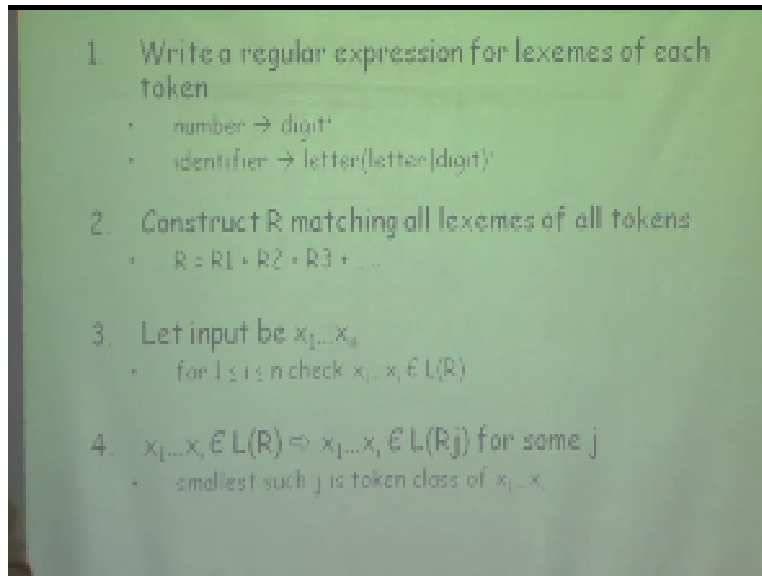


So now once we have regular expressions in specifications we also need to worry about implementation because remember that I can write specifications but somewhere there will be implementation why is what is the problem so regular expressions they describe almost all the languages we deal with should put this by the tokens and obviously as I said in many cases they may not be able to find out what is the exit token and they may pass on information to subsequent phases but remember that these are only specifications and I still need to worry about the implementation part okay.

So question now is that what is it that I am trying to implement what is my input to lexical analyzer lexical analyzer I am saying that our string is and I have a regular expression R and whether this particular string belongs to the language specified by are this is a question I am trying to answer why doing lexical analysis of programmers or in the combine okay but if I say guess solution I mean although this is the basis.

But if I just give an answer which says yes or no okay that is sufficient for our expressions because we are trying to get it information's are subsequent say this so if I just say that yes it belongs to this language okay in between so what do I do with it so what I will say is that I also want to tokenize it I want to generate information in tokenization is really the implementation part so goal is not just to give this answer the sunset will tell me whether it is in a radius token Or not but in more important concepts for me is that I want to partition this in and that is where all this information about maximal munch and ordering of token success I will come into picture and the tools we use okay.

(Refer Slide Time: 25:09)



So let us look at how do we write or what is how the I pass and so how do we token so you want to write regular expressions for lexemes of each token like we have written for numbers and I give devise okay and then what we do is once I have written all these definitions. I now construct our which consists of all these so for each like for number, identifiers and so on it I will have these different regular expressions and using this operator. I am going to construct them large and now the question I want to ask is my input token is a sequence of characters x_1 to x_n okay. And now I want to say that for some value five which is founded by 1 and n I want to check whether $x_1 x_2 \dots x_n$ I belongs to some Errors okay and obviously it will belong to one of these errors so when I say it belongs to errors what that means is that if you belong to one of these and I need to find out okay, so I start because this is now saying that I scan my input from left to right character by character so this is my input and I am looking at now a prefix of this can it be token can it be a that is the question been answer and therefore I say that if $x_1 x_2 \dots x_n$ I belongs to a lot what that means is that actually it belongs to one of the large right and for some value of J and we want to find out that what is that smallest J and that when I say smallest day it is not in size but is in the order okay, so these are the ones which are coming in certain order and therefore when it comes to implementation becomes a important that in which order I specify that regular expressions okay.

So for example yeah coming back to or you already saw in lens okay, suppose I give an order where I say that keywords will come later than identifiers then they will be what will also be matched by then identifier but if I say any five keywords will come first what will you do he will first say that I am going to do this match and only does not matter if this is the longer match

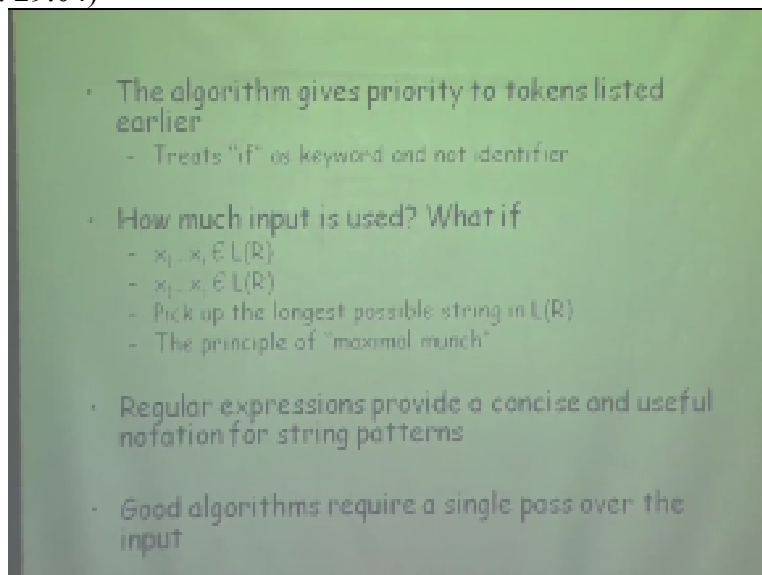
okay, so order becomes ,so different tools are going to give you different way of ordering and suppose.

Too at all suppose you use a C program okay, that one is the order you will have to think worry about what kind of order you going to use that becomes a part so remember that this is not some thing you can just ignore you do not worry about whether you are using tool or whether suppose you are using a C program assembly language program whatever implementation you want to do so I can have these specifications.

But implementation is that is something and you going to worry about and when to say that I have reached the word okay, that is something if you are not cheerful you can just you can just generate sequence of tokens and again which are in very so then once we have identified that it belongs to one of these regular expressions what do I do I remove it from input and then I start tokenizing potato so basically this is saying that I start from in this string.

I find a prefix which becomes a token then I remove that and start looking at the beginning of the next token all the inputs so I just keep going back so every time I go through the citation I am generating Porto okay.

(Refer Slide Time: 29:04)

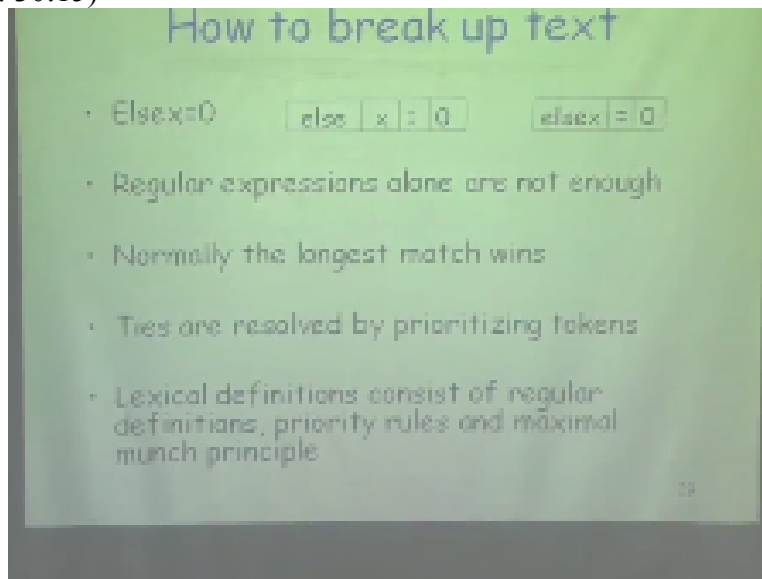


So algorithm normally is priority to tokens Charlie speed ugly so T is a keyword and not an identifier so if I can counter in it first okay, then I am going to say that this is keyword but how much input do I use and that is where we say that I want to do and longest match so normally all lexical and I will say that pick up the longest possible string in the string in the input so here you will say that when I have something like this okay. I am not just stop here but I did say that go all

the way you can conclude and only when you are not able to consume it from this rule so suppose after this I have a less than fine.

Then you know that I have not be able to consume this particular symbol in this spoken because it will not match any of the specifications .I have and then I say this is my word boundary and this will not be mine okay , so that becomes an implementation so regular expressions they are going to provide very concise notation and good algorithm requires a single pass I mean I do not want to have.

(Refer Slide Time: 30:15)



So how do we break up text here are some examples so if I say X is the fine 0 who is I will be able to break it I may be able to say either this or this but maximal much principle will say that because this so normally the longest match is the one which is going to win and is the result right okay, so if you have prefixes and lexical definition basically now consists of regular definitions priority rules and maximal munch principle so here you can see that here if you have clubbed everything.

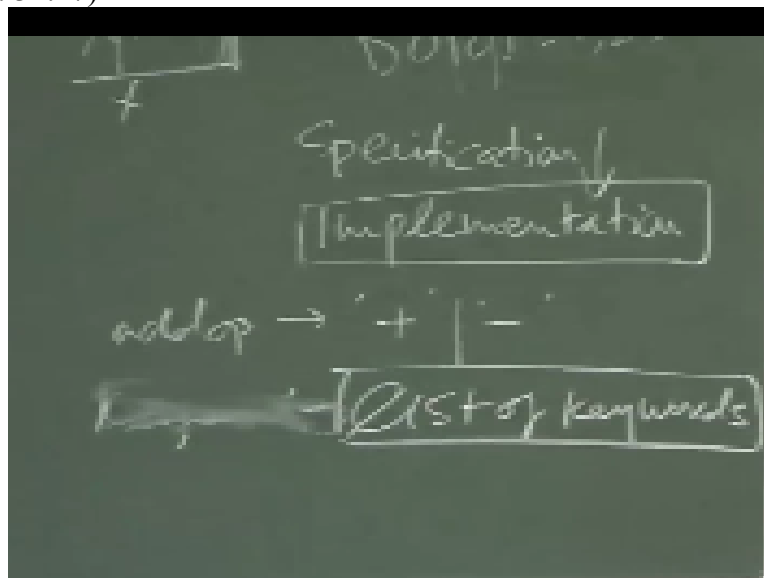
I have not just specification but I also clubbed the implementation would say that order so only thing you have to worry about in sees something like this right so what is the if I do not lose maximal munch then will I break it into this or into this if we take any keyword for example if I if I take this.

I will not reach equal to I start reading from here so I read I then I read F and then I have to take a decision whether I want to read one more F or I stop there so that is part of your implementation now if you say that look keep, looking keep consuming something till you hit about boundary that is saying that go for longest match see my input is staying good where you

specify boundaries is it specifying boundaries for implementation so when I write my regular definitions is based on their specification.

Where I was saying that in the bottom okay, so question is discussion about how do I specify my word boundaries so let me go back okay, so let us look at this yes there is set of specifications. I have four lettered the digit identifiers numbers and I can add more spit one more specification and let me say that which I say is additional and operator.

(Refer Slide Time: 34:17)



Which I will specify as plus or minus this is my complete this can be my complete specification for expressions which consists of fighting fires in numbers and of it now where do I specify one by one . I can also turn one more specification and I can say P word but he word is not really part of specification that we need part of an identifier and then I have a list of keywords so where do I specify that I have certain bones so I understand that part saying where is it specified.

So if it cannot be specified but I have to implement a lexical analyzer okay ,where do I capture this is it part of your specification or part of your implementation if it is popular implementations then we use social principles and make longest match it is not specified anywhere, so somewhere now if I go forward once again and take you to last point is summarized in lexical definitions consists of regular definitions and priority rules and maximal functions a maximal punch is a principle.

I am using here saying that I want to vote for longest night and that is when I can say that keep on reading suppose .I did not have maximal punch and ,I replace that by first match then what will happen so that means your lexical analyzer has to consists of implementation rules as well as the so let us not worry about harmful to the human maximally that is really not the point yet that

point we completed all I am saying at this point of time is that is when you cross off a lexical analyzer it is not sufficient to say that there is a set of regular definitions just go ahead and implement.

Okay you need to specify something more and that is something more is that you say that we always go for longest match or we say that the first specification. I find in that order if I find something which matches are then I use that rather than trying to exhaust everything in that sequence that is part of your implementation, so you cannot ignore this part for implementation of a lexical analyzer that is a only point so not everything is captured just by the definitions this only says what are the very tokens.

But organized that comes from this part that is there all these boundaries are going to get captured this point cleared to everyone the decision so in fact I mean this is a very important point because when you start implementing lexical analyzer in which when you start writing Else x specifications if you are not careful about the order suddenly you will find that token sequence so internally let us assume certain things let us assume that it will token is Else x specifications.

In the order they occur and it will go for maximal munch now if you are not careful about that then you can find that the sequence of tokens if you do not write your specifications correctly in that order then suddenly you will find that sequence of tokens is different so I can take two set of specifications chain the order and my sequence of so it is an implementation. I may have the same set of specifications and if I use a different order of writing that then suddenly you find that my tokenization happens different.

So do we agree on this can you move ahead okay, so what we want to do now is you want to slightly move away from regular definition and I want to introduce a new notation because remember that one thing we were talking about but we said not only you want to use rules for implementation but somewhere. I want to do a manual implementation. I want to write not specifications in regular definitions but I may want to directly write C code for sake off features but when I am writing C code I do not want to be I think C code in a hexagon manner. I want to have some systematic way of specifying or some systematic notation of saying that what are the kind of tokens are and so for that I introduce transition diagrams think okay if I do not worry about properties and so on.

I go for if you petition and you find that and I start talking about transition diagram there is something you have already seen okay, so what we want to do is the regular expressions are some kind of declarative specifications and transition diagram.

(Refer Slide Time: 40:37)

Transition Diagrams




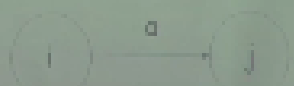
- Regular expressions are declarative specifications
- Transition diagram is an implementation
- A transition diagram consists of
 - An input alphabet belonging to Σ
 - A set of states S
 - A set of transitions $state_i \xrightarrow{\text{input}} state_j$
 - A set of final states F
 - A start state n
- Transition $s1 \xrightarrow{a} s2$ is read:
in state $s1$ on input a go to state $s2$
- If end of input is reached in a final state then accept
- Otherwise, reject

Is really the implementation part so again, I keep talking about so what we do is transition diagram really consists of a set of alphabets which belong to Σ and then it consists of set of states and then, I can have transition from one state to another on certain input and then we have a set of final state and what we do here is that when I say there is a transition from state one who is stateless to a state $s1$ to state $s2$ on a I can say that if I am in state $S1$ and input is H and go to state $S2$.

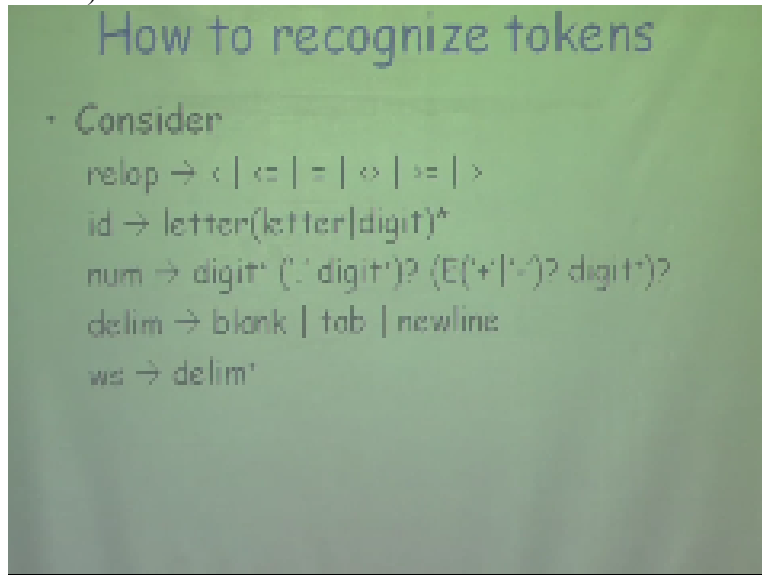
And supposed to be maybe that is on subsequent one yeah so when I reach end or input then we say that we are in the final state is that except it you will find the Quran talked about is something finite state machine and otherwise reject.

(Refer Slide Time: 41:38)

Pictorial notation

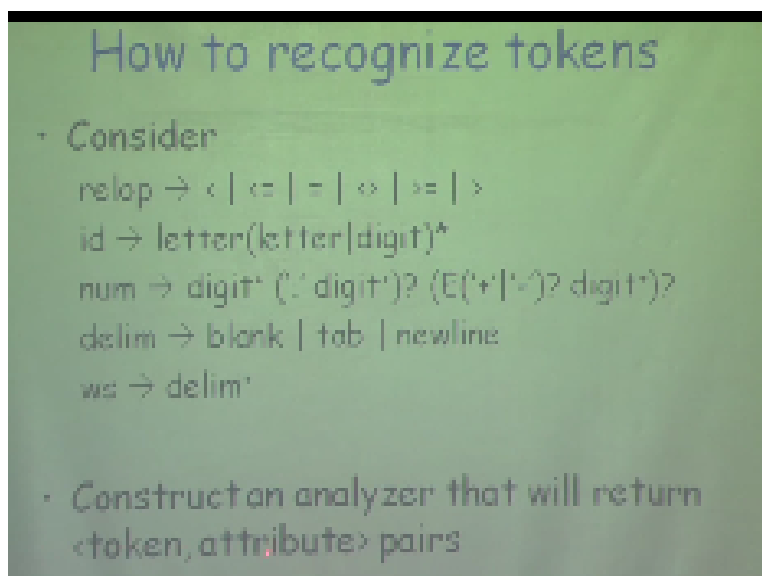
- A state 
- A final state 
- Transition 
- Transition from state i to state j on an input a


So that is on this point if so state. I will use rotation as a circle and final state is going to be to compare in circles transition is going to be an arrow and then I say that transition from state I to state J is going to be captured like this and then transition diagrams becomes very easy way of you specifying my tokens and then going for directly taking me to the implementation.
(Refer Slide Time: 42:02)

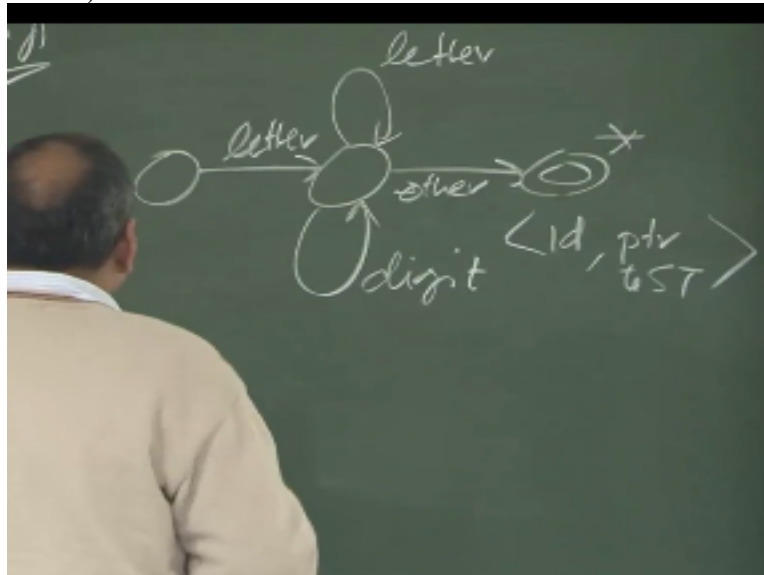


Okay so how do we recognize focus what we try to do is we try to now develop transition diagrams so these kinds of specifications vary. And I have ended my language climb saying that I have these relational operations I have identifiers so I have numbers are millimeters which are blind caps over a new line and then, I say I have white spaces which are one or more occurrences of a delimiter okay.

(Refer Slide Time: 42:29)



And I want to know develop transition diagrams so what we want to do is we want to construct now a lexical analyzer which will give me this token attribute pair and here I want to go for manual implementation so I do not want to use regular definitions .I want to sum so let us take let us start with something and start sort of developing diagrams for this and this background will end this diagram will then start capturing all the implementation which is done okay so if I want to deal with identifiers if I am in some state and so if I try to develop.
(Refer Slide Time: 43:14)



Now this condition diagram for identifier time in some states what the first character is. I can see first connector I can see is only a letter right so if I see in this step. I can only see the letter invalid that he takes me to the next state and then in this page. I can continue to see either a letter or a digit okay, so I can have this level which will say I can have a letter or I can see digit and then I can go to a final state by seeing other let me use okay and what this says is that anything else which is not a letter or digit will take me to this state.

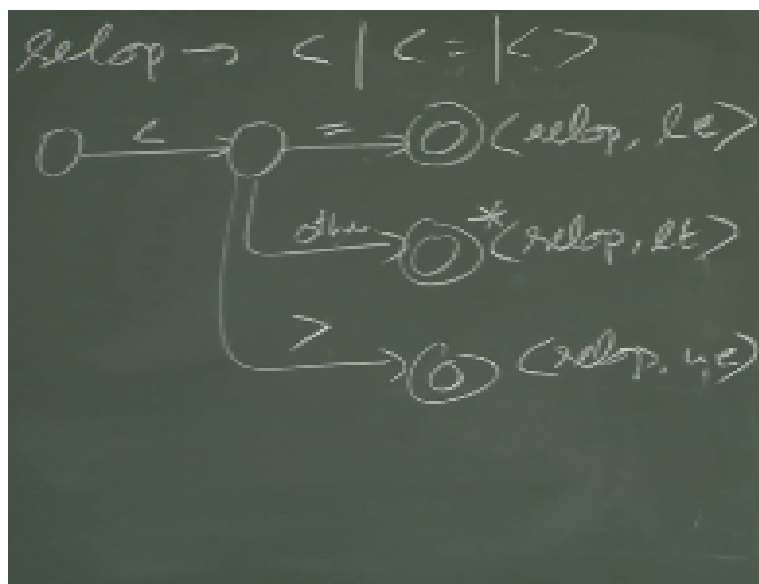
Which will say final state and in this final state now I want to do something more because I am talking of implementation of what is what all the things I am doing in this final state I want to now return a token and let you do it so what is the token here. I am going to return it is an identifier right so I will say that my token is ID and what is the attribute here so if this texting has been identified okay, then I am going to put this information in the symbol table or it was already.

In the symbol table then I just need to have a pointer but it should be pointed to simple table corresponding to this Lexal but then I need to do something more and what is that something more so I have identified. I have written this pair but what else do I do so we have consumed one

extra character okay. I must specify somewhere that this extra character must be returned back into the income streams so I just put a rotation here so you can see that what I am talking about is non-implementation rotation therefore.

I did not say that this finite machine and so whether it is capturing all the implementation detail you say yeah so here I say that this will remind you of saying that the last character which you have consumed here must be returned back to the inputs ,so this becomes your specification for implementation off if I try to do let us say for a relational operation now what will happen in relational operation.

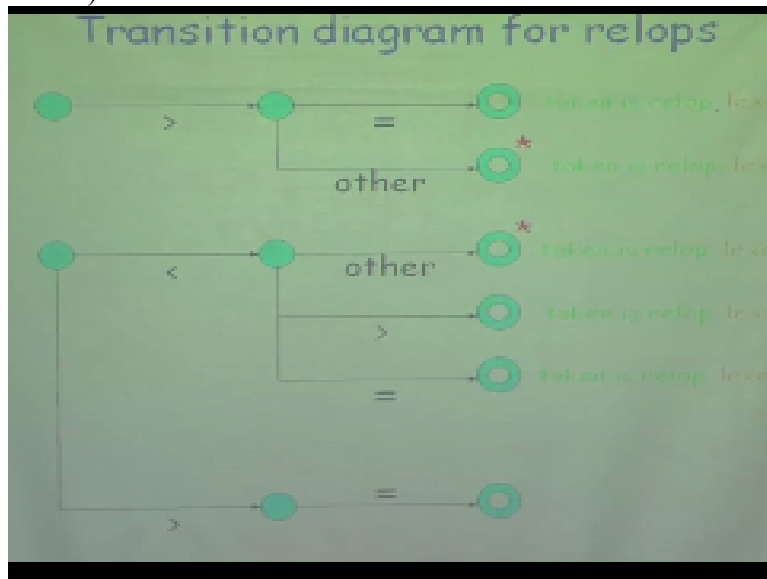
(Refer Slide Time: 45:43)



So I being some start State okay, so let us look at for the just possible so if I just say my relops e so in this page what can I see so does not matter whether , I am trying to match this token or this token. I will always be less than and that will take me to a new state and this new state either .I can see an equal or I can see something else now if I see equal okay, do I need to look at more now suppose I do not have anything more in my specifications just that I am saying that this what I match and therefore I can reach a final stage and then.

I can say that return now read off and let us say less than equal to a but I do not have to return anything but if I reach this as a final state then I am saying that I have matched realm oh I have less than and then I must return and then if I just am bitch the whole thing if I say that I can see say it let us say this also okay ,then what will happen so in this state I could have had other label which would have said and then as soon as I say greater than sign definition of these other changes so this other is all symbols.

Which are not so all symbols minus symbols on all of the labels so this definition of other if I did not have this edge then this was Σ minus equal but as soon as I add this edge they say this is Σ minus equal and then this will immediately say that what I have seen here, so this is not equal to so now I say that I am saying that this is this and that will give me the full specification okay so once we have this I can once you have understood this part.
 (Refer Slide Time: 48:20)

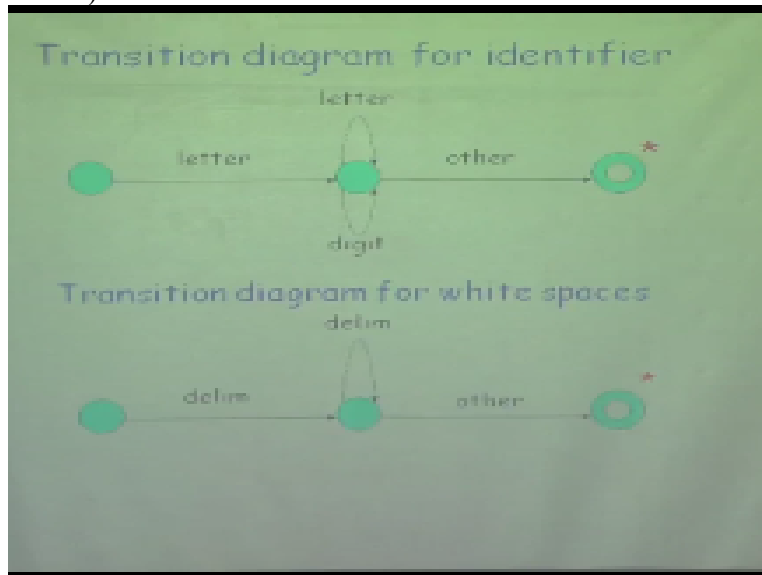


I can take you to transition diagram so for relops okay, this can be one part of the transition diagram which says greater than equal and this will just say that, I token is real of them is greater than equal to or I can also say that the spoken is lexeme could be this and many times what you will find is that it may have additional specification by saying this is greater than okay but as soon as I say this is greater than equal that is sufficient to say and other will take me to this specification but in this case.

I will also have to return something with this right so let me just finish this Foundation diagram and then we can break and for the second part if I am now trying to capture everything if I say this is less than then immediately. I know that I have reached the final state and by because whenever I consume other I must return it to the input stream and when I say this is less than equal then it is saying that I am reaching now this final state.

But I am not returning anything but if I say greater than then I am reaching another final state and again I am not returning anything okay, and if I much if I takes all the relational operations then I in this state in the start state. I can not only see less than but I can also see greater than which really is this transition diagram so it will get captured here okay it and in the start state I can also see equal which is here.

(Refer Slide Time: 49:49)



Sorry just too fast yeah, so if I see equal then immediately I know that I am reaching in this particular state and I am just not in equal so you can see that this becomes now your implementation specification for relational operation so next we will finish today and we will meet in the afternoon at 4.30 okay.

Acknowledgment

Ministry of Human Resources & Development

Prof. Phalguni Gupta

Co-ordinator, NPTEL IIT Kanpur

Satyaki Roy

Co Co-ordinator, NPTEL IIT Kanpur

Camera

Ram Chandra

Dilip Tripathi

Padam Shukla

Manoj Shrivastava

Sanjay Mishra

Editing

Ashish Singh

Badal Pradhan

Tapobrata Das

Shubham Rawat

Shikha Gupta

Pradeep Kumar

K.K Mishra

Jai Singh

Sweety Kanaujia

Aradhana Singh

Sweta

Preeti Sachan
Ashutosh Gairola
Dilip Katiyar
Ashutosh Kumar
Light& Sound
Sharwan
Hari Ram

Production Crew

Bhadra Rao
Puneet Kumar Bajpai
Priyanka Singh

Office

Lalty Dutta
Ajay Kanaujia
Shivendra Kumar Tiwari
Saurabh Shukla

Direction

Sanjay Pal

Production Manager

Bharat Lal

an IIT Kanpur Production

@Copyright reserved