

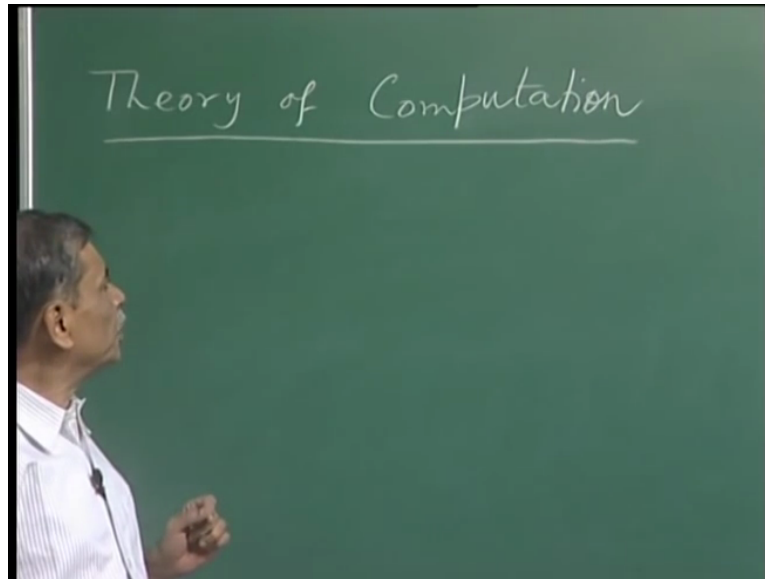
Theory of Computation
Prof. Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Lecture 1

What is theory of Computation?

Set of membership problem, basic notions like alphabet, strings, formal languages.

(Refer Slide Time: 0:23)



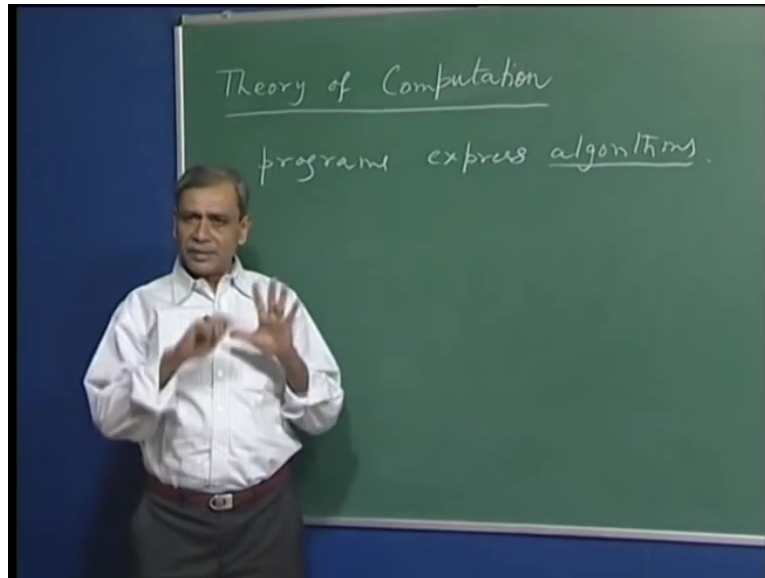
Name of this course is theory of computation and let us spend a little time to understand the scope of this course. Now we have the word computation in the title and it is saying that ever course is what? The theory of this entity called computation. Now when we talk of computation, today clearly what we understand as computation is that you write some programs and you run the programs and computers and then in that way you carry out a computation.

So fundamental activity behind computation or the way computation is carried out by executing programs, so what are programs? Let me let me write this term which we are all familiar with programs and anyway we can see that our theory is about programs, right? There can be various theories I need to now say clearly, specifically what we mean by our theory that we will discuss here.

But again if we think of the word program we know there is something fundamental behind this term even more fundamental because after all a program is nothing but an algorithm expressed in a programming language. So we can say programs expressed algorithms and you

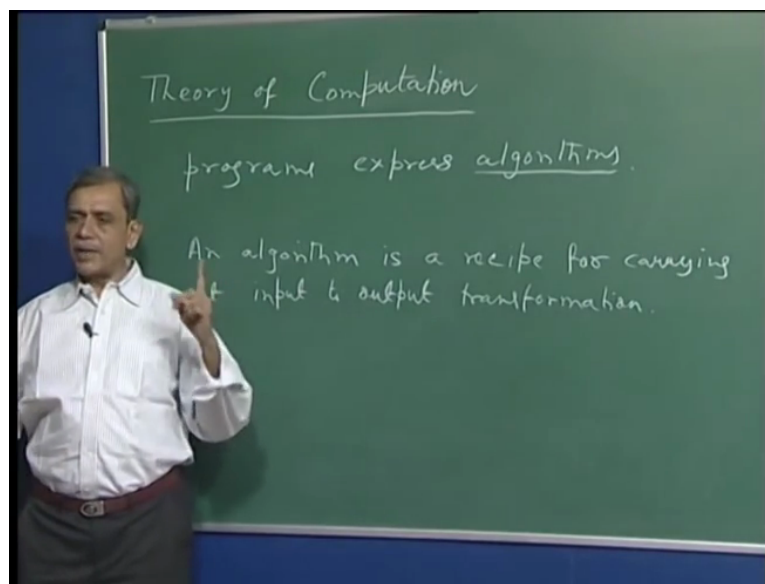
see already we are making an abstraction in this, that we are saying that programs express algorithms, so more fundamental notion than programs is the notion of an algorithm.

(Refer Slide Time: 2:47)



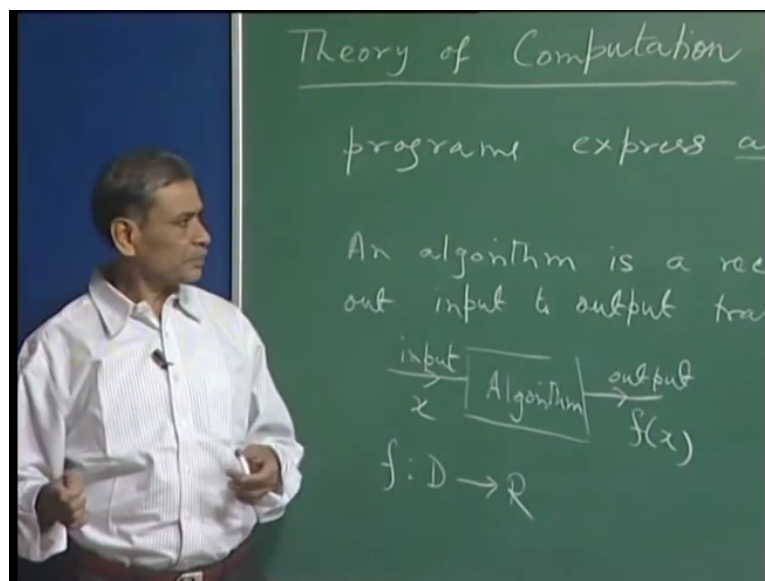
So going back again what we can see? That our theory is about theory of algorithms, right? Theory of computation, theory of programs and theory of algorithms all more or less denote the same thing in our context. What are algorithms for? We all know that but let us let us let us announce clearly, we know that algorithm what it does, is it computes or it gives us a recipe for computing and input output transformation.

(Refer Slide Time: 3:27)



So let us again write this an algorithm is a recipe for carrying out input to output transformation this seems obvious enough that when I have a program, it is executing an algorithm and what is that algorithm? That algorithm is a finite, effective, step-by-step process or description of a process which we are calling a recipe more familiar term for carrying out transformation of the given input to an output.

(Refer Slide Time: 4:56)

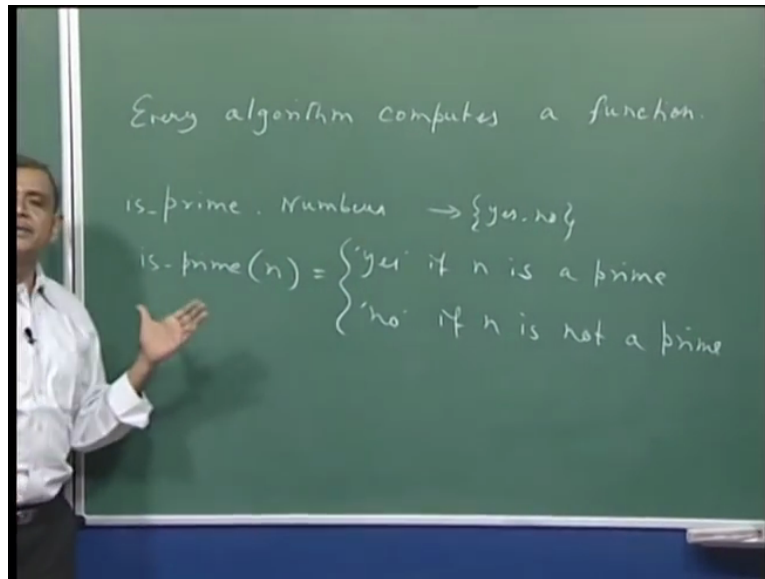


Another way of saying this would be that an algorithm tells us how to compute a function? So you see if I write like this recipe for carrying out input to output transformation. So in picture it will be like this is an algorithm and as we know that it tells me for a given input how to get it out but then we can see this as telling me what is the output for, let us say the input X ?

If I call that output, if I term that output for X as f of x then we are we are dealing with an even more fundamental familiar concept is that of a function, right? So as you know a function f is a mapping from some domain to some range and in this case the domain is this domain of all input and the range is the range of all outputs for a specific function, right? You may have an intuitive understanding of a function, right?

You may have an idea what the function should compute? In other words given an x what effect should be but that is not enough for actually obtaining given an X , the value fx for that what we say is that we have to give an algorithm to compute that function f . So we can say that an algorithm computes a function again that is kind of very straightforward simple understanding, right?

(Refer Slide Time: 7:18)



Every algorithm computes a function, right? So let us write this here every algorithm computes or I should have said tells us how to compute a function? The distinction should be clear even algorithm and function we all understand, function is an abstract notion which tells me that here is this mapping between input and output domain and range and algorithm is telling me how to obtain the output for a specific input, right?

In other words we are using the word computes in that sense. Now I can say basic goal of our theory is to figure out for what functions we can have algorithms? Now that might sound a little unfamiliar we may think that if we can dream of function, if we can define a function, if we can in some way tell somebody else about a function then that itself should be an algorithm but really that is not the case.

For example, let me let me provide a very simple illustration. Consider this function which is is-prime, right? This takes a number to let us say to my domain is the set of numbers and the range is yes or no and by that what I mean is, supposing I say is-prime and then apply it on a number then it is going to be, the output is going to be or the answer the value of this function on the on the argument n is going to be either yes or no.

And now let me define when I want the answer to BS and when I want the answer to be no. So let us see it should be yes if n is a prime, the number n you have given as argument is a prime and it should be no if n is not a prime, right? So you have defined this function but you see clearly we are not saying, this definition itself is not telling me given an n how to actually

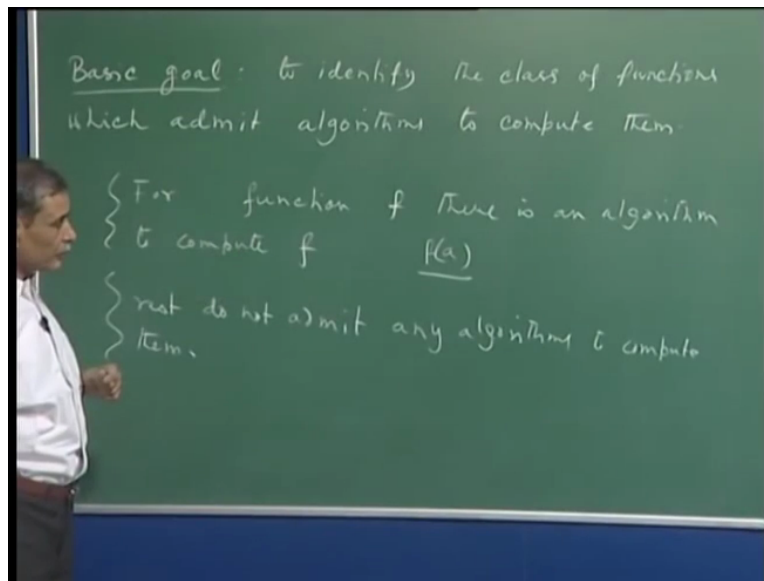
come up with the answer with the right answer and that would be as you know and you have surely written a program to compute this function.

You will have to give an algorithm which will tell me how to give an argument n which is a number n , how to figure out whether it is a prime or not? And that how to should be expressed in terms of simple operations which can be carried out on a computer, right? So these are all we know in other words we would like to write a program to compute this much. So you see that underline, that program, that algorithm is completely a function.

So coming back what we are trying to see? That it may be possible to define a function but the definition of the function does not immediately point out in all cases to an algorithm to compute that much. If that is the case then at least you can now see that there is a possibility that I may be able to define a function I may be able to describe what the output should be without having an idea how to obtain the correct answer?

Although you have not possibly encountered such situations in our programming experience but it might surprise you when I tell you that actually it is a fact that for most functions there are no algorithms to compute. In other words if you think of the class of all functions then only a tiny fraction only a tiny subset of these functions admit algorithms to compute them and this is something which we will be able to prove which we will be able to demonstrate in our theory. In other words let me say of a primary goal of our theory is going to be to figure out which functions can admit or will admit algorithms to compute them.

(Refer Slide Time: 13:13)



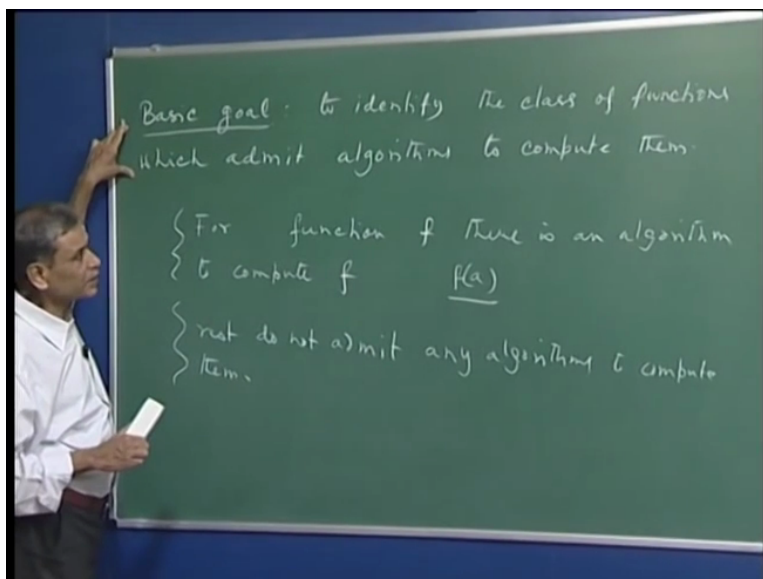
Basic goal of our course that we are going to go through in the series of lectures may say as it is to identify the class of functions which admit algorithms to compute. So what we are saying is that there are some functions which are of the kind that for function f there is an algorithm to compute f , right? And there are some other functions rest do not admit any algorithm to compute them.

In this class of functions that if you give me an input a then I will be able to tell you by carrying out by computing the corresponding algorithm for f , what the value of the function will be on the argument a whatever be the algorithm? But in this case it may be that for some inputs I will be able to give but there may be inputs for which I will not be able to tell you what the output is.

In other words I will not have an algorithm in general to compute that function and as I said that most of the functions, unfortunately if you like they fall in this class that they do not admit any algorithm to compute them.

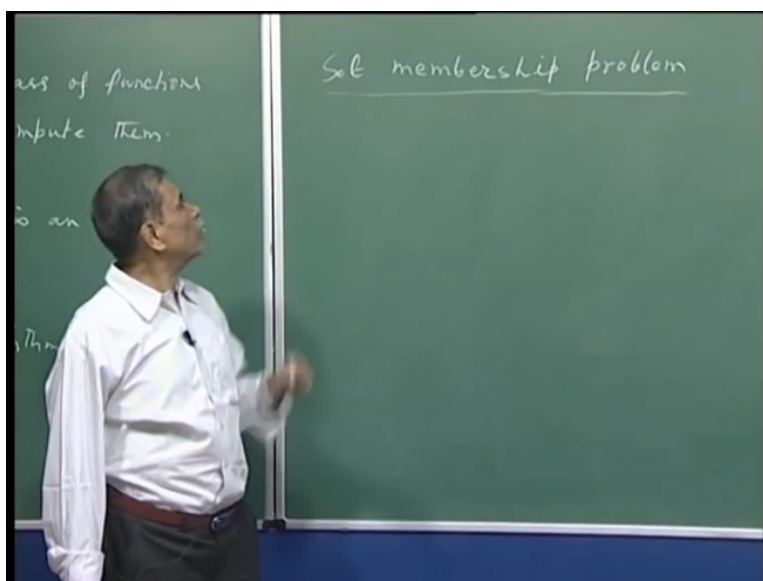
So at the end of the course in principal you should be able to not only have examples which are very clear and which are in a way important functions for which we will not have any algorithm we should be able to prove that such is the case with those functions and in principal again we will have the knowledge of the techniques to demonstrate that a function does not admit any algorithm to compute them and that as I said to be at the very end of this course.

(Refer Slide Time: 16:40)



Now the way we look at this problem although this is the fundamental thing, this is our basic goal is to identify the class of functions which admit algorithms to compute them but you will see in the course actually we will not talk about functions instead we will talk about a problem which is kind of related and I will have to show you the relation and that problem is called membership question. You know given a set and given an element whether that element is a member of that, so this is a question that will engage us for most of the course and let me clearly explain what that question is.

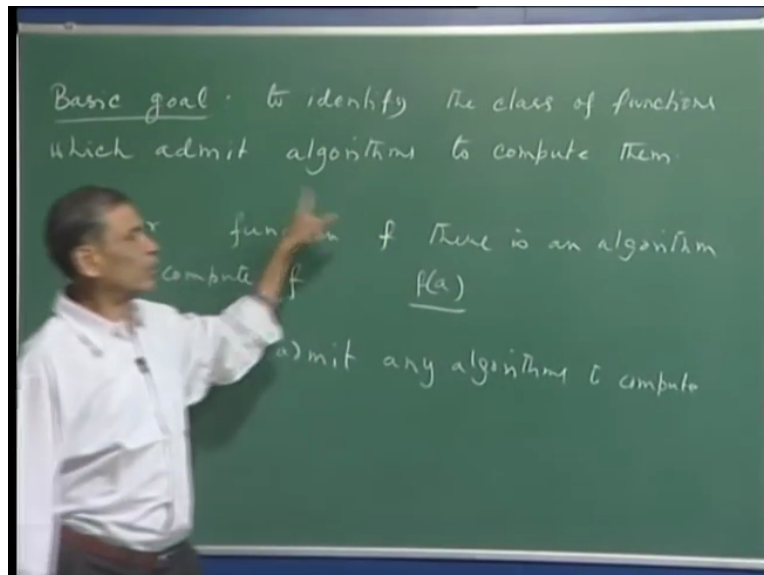
(Refer Slide Time: 17:31)



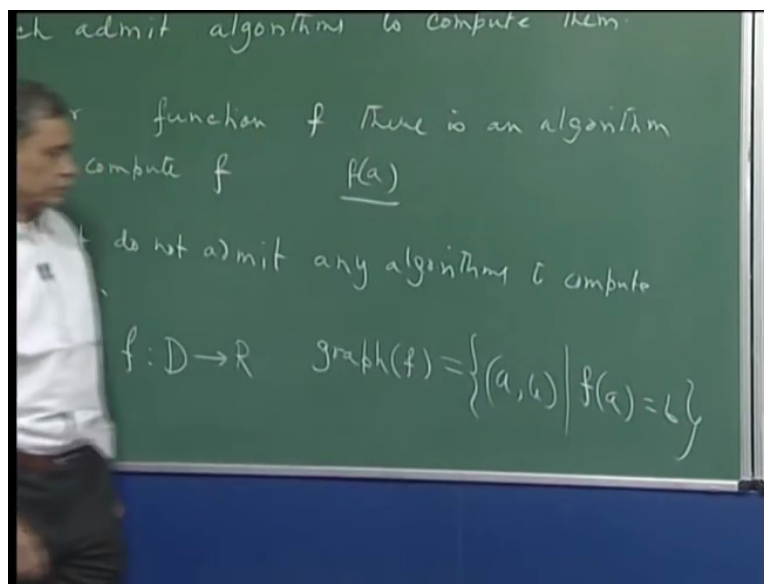
So let me understand let us understand what this problem is? Set membership problem, actually very simple to state, it is a very general problem that you know you have some set S and our problem is that given any a as input to decide if a is a member is an element of the set S , so very simple, right? There is a set, in some way we understand what is that set? And now you give me some element a and we would like to know whether this element belongs to the set S or not.

Now as I said all the time in this course will really be talking about this problem, what we will do is, we will be able to show various kinds of algorithms not the way you study in algorithm's course but through various models of computation the classes of set for which we can carry out this problem.

(Refer Slide Time: 19:14)

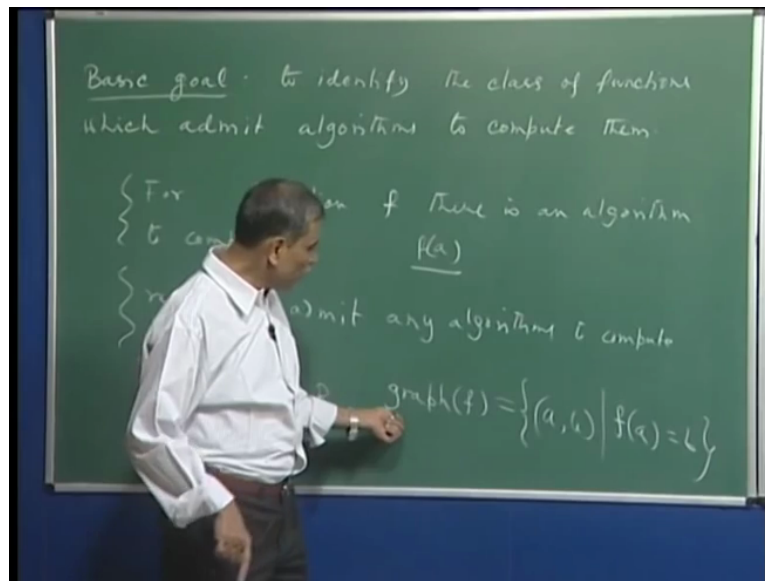


(Refer Slide Time: 19:37)

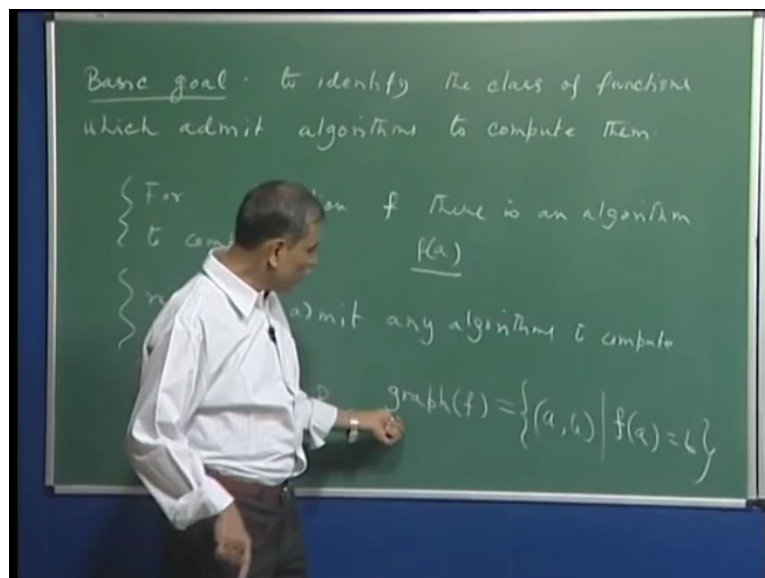


Now what is the connection between set membership problem and the basic goal that I had talked of here? You see if you think of functions f then remember that we said that function F , notationally we always write like this that a function f is a mapping from some domain to some range R . Now you might know that there is a very natural set which is associated with any function and that set is called the graph of f , right?

(Refer Slide Time: 20:55)

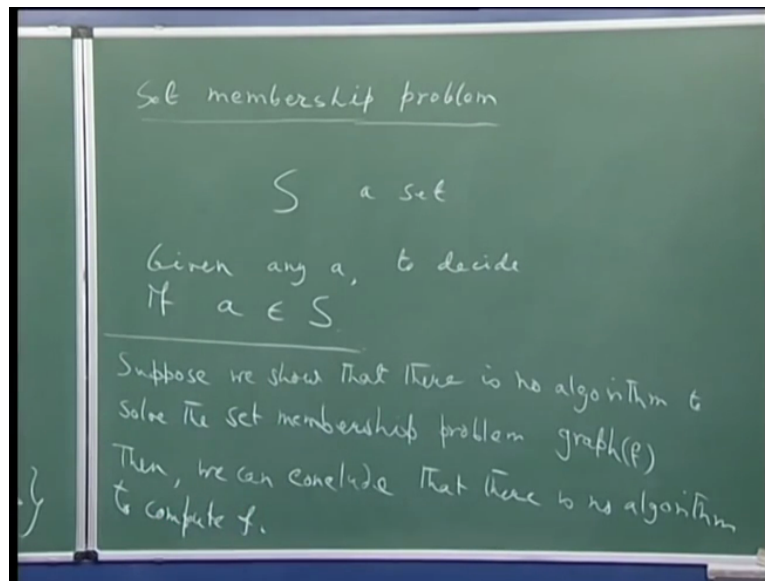


(Refer Slide Time: 20:58)

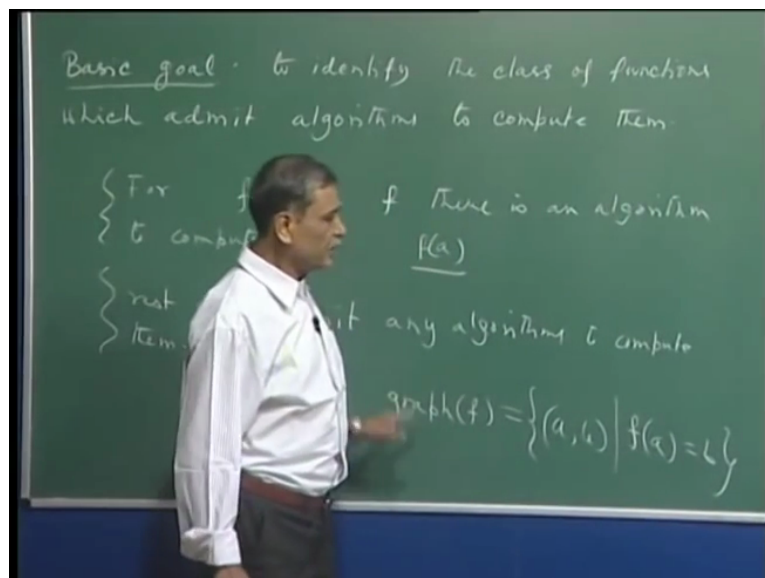


And that graph of f is a set and what is that set? That set is a set of tuples a, b such that f of a is equal to b . Now the point is this that with every function f the we can associate such a graph and now suppose we cannot decide we cannot give an algorithm to compute or to carry out or to solve the set membership problem of graph of f .

(Refer Slide Time: 21:08)



(Refer Slide Time: 21:56)

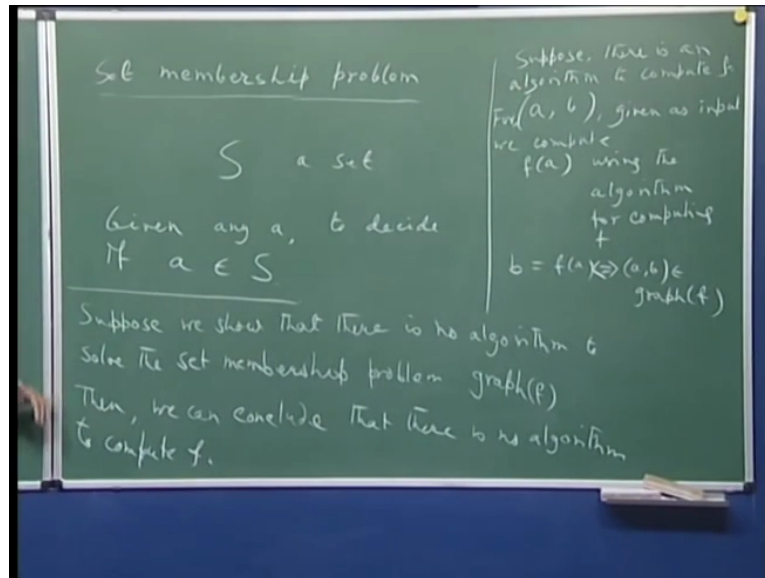


So let us write it down, suppose we show that there is no algorithm to solve the set membership problem for the set graph of f , now but this is a set and as we said this is this particular set is defined like this. Suppose we show that there is no algorithm to solve the set membership problem for the set graph of f then we can conclude that there is no algorithm to compute f also.

So what I am saying is that if we show that there is no algorithm to solve the set membership problem of $\text{graph } f$ then there is no algorithm to compute the function f itself and this is quite easy to see and we can very easily prove the equivalent contrapositive statement and which

will see that if there is an algorithm to compute f then there is an algorithm to solve set membership problem of graph f .

(Refer Slide Time: 23:39)



So if you have an algorithm to compute f , right? Then if somebody gives you this tuple a, b now and you have an algorithm to compute f then what you can do is look at the first argument and compute f of a using the algorithm for computing f , let me write this suppose there is an algorithm to compute f then for a, b given as input we compute $f(a)$ using the algorithm for computing f , right?

Somebody has given you the algorithm to compute f , so you use that algorithm to compute f of a , a is what? The first argument of this tuple and now supposing the is f of a then of course a, b this implies actually b equal to f of a if and only if a, b is a member of graph of f , right? So therefore using the algorithm for computing f I can solve the set membership problem, the set membership problem was given a, b check whether this tuple is a member of graph of f .

This tuple is going to be a member of graph of f , if and only if $f(a)$ is equal to b and now I have an algorithm to compute f and therefore clearly I take the first argument I compute f of a check whether b is equal to f of a and then I can answer whether a, b is a member of the set graph of x . So therefore this statement we have proved it is so easy, right?

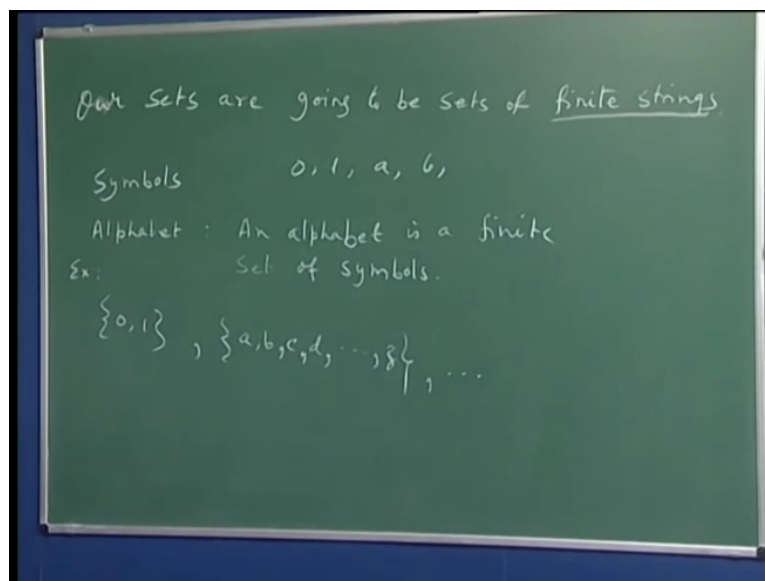
Now if I can show therefore that there is no algorithm for computing f , there is no algorithm to solve the set membership problem of graph f then I have shown that there is no

algorithm to compute f and this is the reason you see that a function which is not computable we will be able to get to that fact while looking at the graph of that function and grouping that graph of f is something is a kind of set, is set for which I have no algorithm to solve the set membership problem.

Now what, what does it by? You know instead of functions then I can talk only about sets and sets are more fundamental and in a way simpler objects and therefore the theory at least notationally becomes simpler although it manages to show existence of functions for which we have no algorithms to compute those functions through the study of some kind of sets and this is the reason most of the, in fact entire course will concentrate on sets and their membership problem rather than functions.

In fact, although that is the basic goal in our mind because we would like to talk about programs and whether or not we can have a program to compute certain things. You will see in fact that this problem is not something we tackle in its and it is generality, what I mean is that the kind of sets for which we will consider set membership problem, the sets are going to be very very special kind of sets and what are the sets for which we will consider set membership problem?

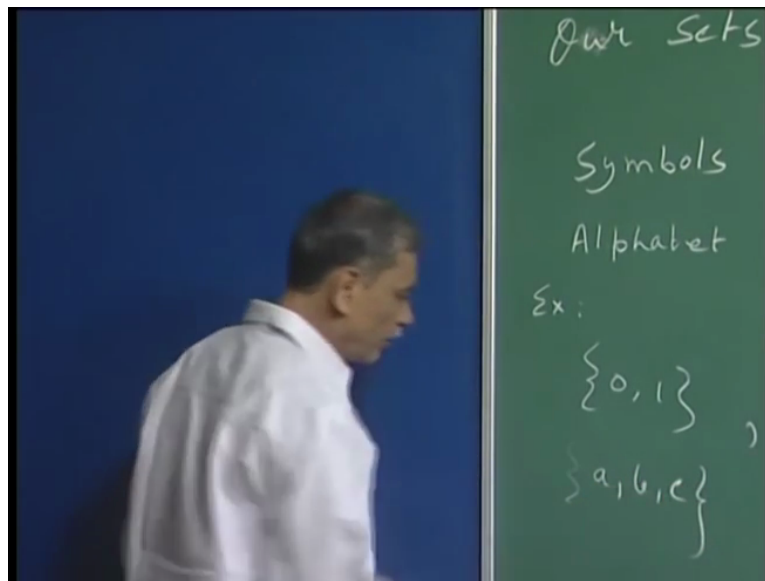
(Refer Slide Time: 28:57)



If these sets are, let me write it. Our sets are going to be sets of finite strings. Let me explain these terms and to talk of strings I need to first talk of alphabets and to talk of alphabet I need to talk of symbols. Now, so let me first use the term symbol, what is a symbol? We are not going to define it only thing we would like to say about symbols that they are as you know them for example 0, 1 these are symbols, right?

Similarly a, b these are symbols, right? And another term that we are going to use is called alphabet. So an alphabet is a finite set of symbols. As an example this is an alphabet, right? There are 2 symbols 0 and 1, the set comprising of these 2 symbols is an alphabet, also another example could be a, b, c, d of the way to z and you can of course give many other examples of finite sets which are made up of or which comprise of symbols.

(Refer Slide Time: 31:33)

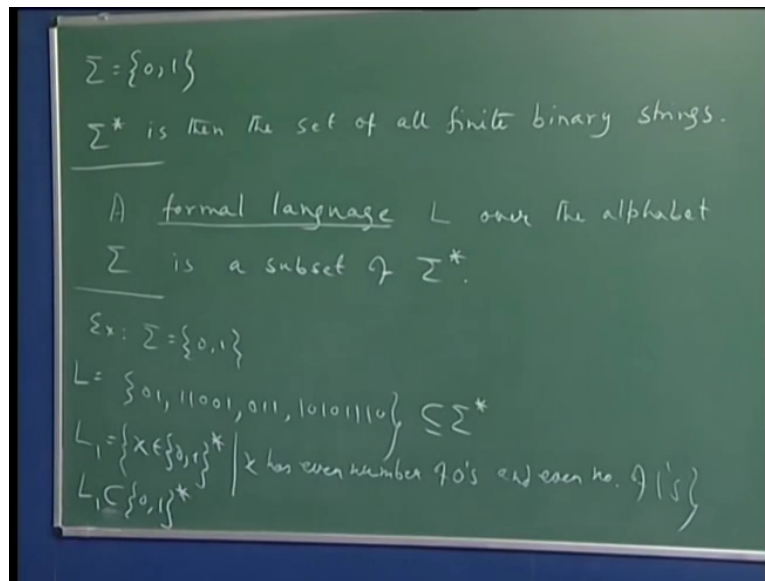


And now let us just consider this set a, b, c to this is an alphabet because these consist of, it is 3 members 3 symbols a, b, c and again another familiar term is a string over these symbols. So what is a string? String is we just write an ordered way, we write this symbols one after another after some point. So let us say b, a, a, b, c this is a string over this alphabet, so we can we can formally define all these more carefully but this notion is so simple you get it immediately that this is a string in which the first symbol is b , second symbol is a , so there is a notion of ordering.

We can talk of first, second, third, fourth, fifth and also there is a notion of length because this has, you can see its length is 5 because it has 5 symbols, right? And this is also a finite stream they could have been in finite strings over the same alphabet but we will restrict ourselves to finite strings over an alphabet.

Now suppose Σ is an alphabet then Σ^* denotes the set of all finite strings over Σ , okay. Now supposing my Σ is $0, 1$ suppose as example I take Σ is $0, 1$ then what is Σ^* ? It is not difficult to see, so what are the strings over this alphabet they are going to be? Binary strings and we said Σ^* denotes the set of all finite strings over Σ and this is the binary alphabet, so therefore Σ^* in this case is going to be the set of all finite binary strings.

(Refer Slide Time: 34:26)



So as we are saying that consider the binary alphabet which consist of just 2 symbols 0 and 1, in that case the Sigma star is going to be the set of all finite binary strings, right? By this, what do I mean to finite binary strings? You know we do not allow in Sigma star to have an infinitely long string but all strings whose lengths finite they will be members of Sigma star and those are the finite binary strings.

So there is really no upper bound on the length of strings which we have in Sigma star but at the same time every string, any particular string that we take in Sigma star that has a finite length, is this notion clear? That this is an infinite set and all its members are you know are finite lengths like set of numbers set of integers that is an infinite set each integer itself is some member of finite object.

It is let us say 10,000 it will be 1000,000,000.00 it could be 100,000,000 so then numbers can be arbitrarily large. Similarly here the strings can be arbitrarily large but each string in Sigma star is finite, right? So now that is one very important notion that we are going to introduce that notion is that of the formal language. A formal language L over the alphabet Σ is subset of Sigma star, okay.

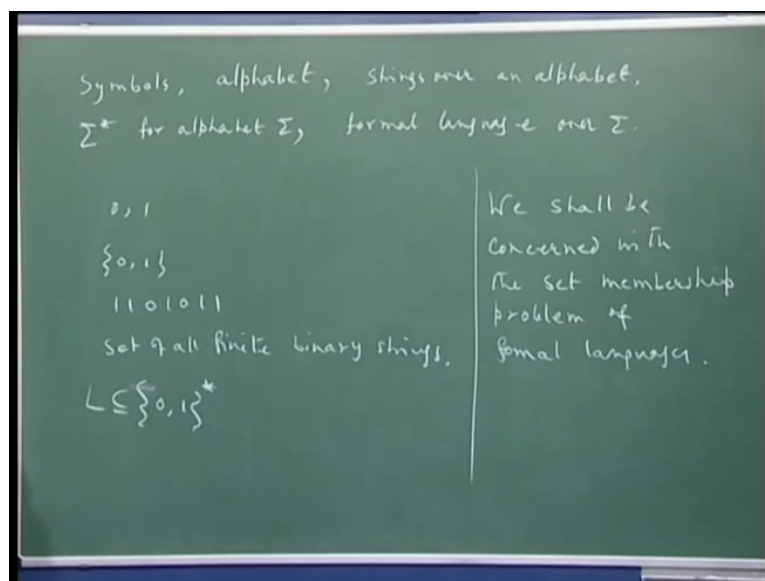
This is a formal language L over the alphabet Σ is a subset of Sigma star, right? Sigma star recall it has the set of all binary strings over the alphabet Σ and you take some of them, possibly an infinitely many of them leaving may be some out is and that constitutes a formal language over the alphabet Σ . So it is clear that for this alphabet for Σ is

again binary alphabet 0 1, 1 1 0 0 1, 01 1, 1 0 101 1 1 0, right? This is a set of strings over the alphabet 0, 1 and clearly this is a subset of this set Σ^* , so this is a language L .

So we can say that this is an example of a binary language because all its strings are over the symbol 0, 1 and this is of course the finite language but more interesting would be the example let us say L_1 which is x is in 0, 1 this is an alphabet. Now I am putting the star over it, so I mean x is a binary string such that x has even number of 0's and even number of 1's. How many such strings are there? Clearly infinitely (∞) (39:04).

At the same time not all binary strings are in this language L_1 , so L_1 is an example of a language over 0,1 which is infinite, right? But which is not the entire Σ^* . So this language L_1 is a proper subset of, we can write like this is a proper subset of Σ^* . So we have introduced number of terms, let me just enumerate them and briefly explain once more what each are, each of these symbols is.

(Refer Slide Time: 40:00)



First of all we talked of symbols then we talked of alphabet, right? Then we talked of strings over an alphabet then we talked of Σ^* for alphabet Σ and finally we talked of formal language over Σ , very quickly symbols are like 0, 1 these are symbols, alphabet is set of symbols, so this is an alphabet string over an alphabet over this alphabet 1 1 0 101 1 this is a string, right?

Sigma star over this alphabet is the set of all finite binary strings and a formal language over this alphabet is just a subset of Sigma star, L is a subset of, for this particular alphabet 0, 1. So therefore L such a formal language is also a set which has a number of finite strings in general over the alphabet that we are dealing with in this case, in this example at least that alphabet was the binary alphabet.

Now the connection with what we were saying before is this, that we shall be concerned with the set membership problem of formal languages. So now therefore our problem is going to be somebody has some formal language L in mind and the problem therefore is very concrete they give me string and I have to determine whether that string is a member of the formal language.

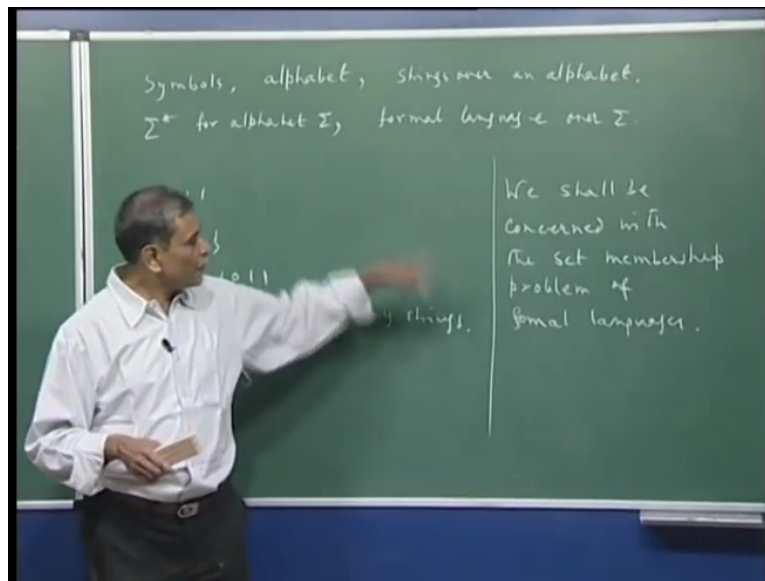
Now very briefly although we had said in terms of the graph of the function it can be seen as a set but then what is the justification of talking about restricting ourselves only to set membership problems of languages and that is because the idea is really very simple, you see when you write a program, right? Then that program takes some input and that input is, is what?

Is actually a string because that input you have you need to key it in through the keyboard or maybe it is coming through a file whatever it is you can think of that input to be a string over some alphabet and the output is again in general is of course not yes or no as in the set membership problem but here we appeal what we have said earlier? Because our goal is to show that something cannot be done by programs.

And so if that is a function then if we can show that the corresponding graph which is going to be in this case going to be basically strings you know basically graphs, those graphs of those functions that we are interested in, since our inputs are going to be strings, outputs are going to be strings, so these are going to be pairs of strings pairs of strings themselves can be seen as a string and therefore it is really a set membership problem where the sets are really sets of strings and these strings are finite strings, right?

I mean you never or it does not make sense to say that my input is a string which is infinitely long because such an input you can never even provide to the program completely for that program, your program to work on that input. So inputs are always finite and in fact there are always finite strings in the programming context.

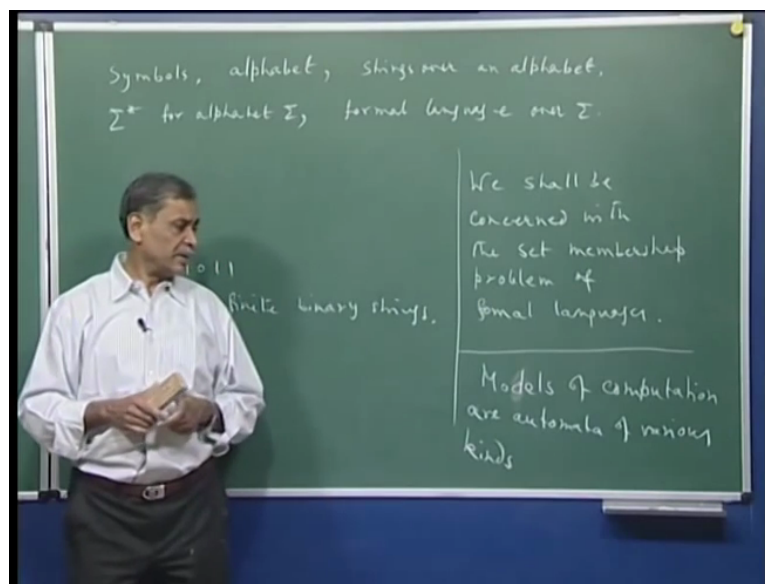
(Refer Slide Time: 45:50)



So therefore this restriction is not something which is very restrictive, right? Although it seems the great restriction from functions we are coming to set membership and then again we are saying that we are not interested in all kinds of sets but only sets which are just sets of strings over finite alphabet. Now that really is the basic issue in this course that you know ultimately of course we will be talking about you know this question of whether certain set membership problem admits an algorithm or not.

But we will come to that goal in a series of steps if you like. So what we are going to do? Is we are going to do we are going to invert the problem in some sense.

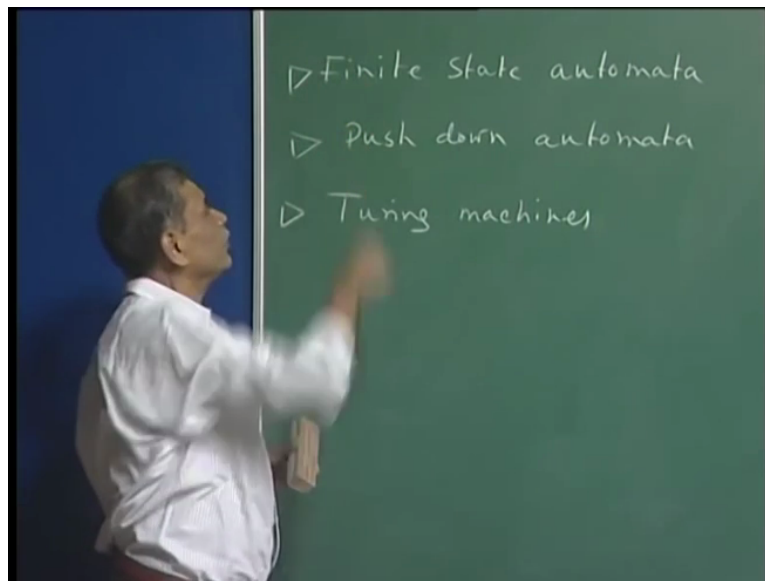
(Refer Slide Time: 47:09)



We will think in terms of models of computation, let me write this down, right? And that means some abstract way we are seeing we will describe a class of algorithms and the that abstract we in fact going to be by specifying what are called automata? See these are again terms which will be much clearer when we see really examples of when we deal with them.

So our models of computations, let me just say this for the time being are automata of various kinds. So it will define a class of automata and then we will ask the question what kinds of set membership problem this class of automata can solve? The kinds of automata that we are going to see.

(Refer Slide Time: 48:41)



We will start with a very simple class of automata they are called finite state automata. Now even here there will be various crimes we will see later the deterministic not deterministic and then we will study what kinds of languages for which this model of computation can solve their set membership problem? After which and this is going to take at least you know 15 to 20 lectures and then we will talk about in terms of automata it is going to be pushed down automata.

And again we will study the classes of languages or which pushed down automata will be adequate to solve their membership problems then finally we will look at the most general class of automata which is possible, you might have heard of this term, you have heard Turing machines and as we go along there will be many other things you know we will study and for example we will study the notion of grammar and here the study regular grammar, context free grammar, we will briefly mentioned what are context sensitive grammar?

And then finally the unrestricted grammar and there will be some correspondence with these things that we are going to study and it is at this point where I said at the almost when we come to the end of the course or in the last part of the course we are going to encounter set membership problems for which there will be no algorithms to solve, we will stop here and in the next lecture onwards will start with finite state automata (()) (51:16) and we will we will be dealing with them for quite some time.