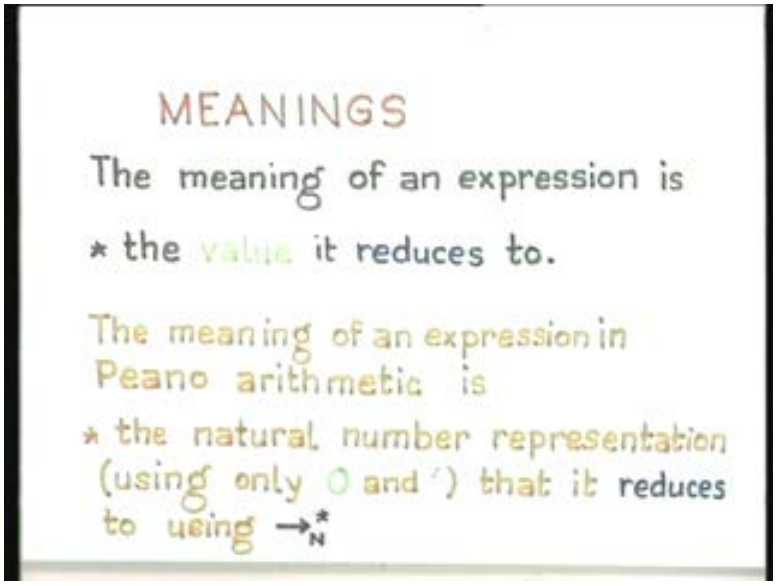
Principles of Programming Languages

Dr. S. Arun Kumar
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture - 30
Normal Forms

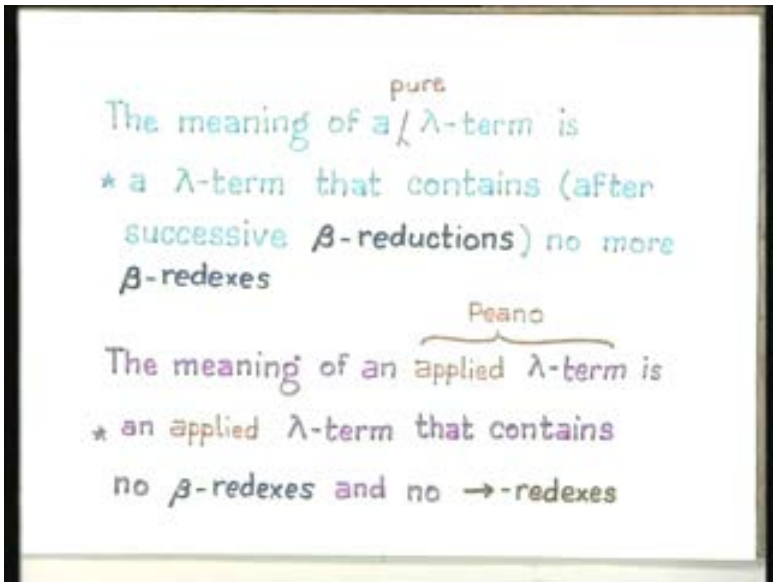
Welcome to lecture 30 so let me just briefly go through what we have done and continue it in normal forms. One of the things we did in any kind of computation is that we expressed the meaning of an expression as a value it reduces to.

(Refer Slide Time: 00:56)



The meaning of an expression in Peano Arithmetic is just some representation of a number which is the value of an expression in the lambda calculus is a lambda term that contains no more beta redexes. In an applied lamdba terms then it is something that does not contain either any beta redexes or any peano redexes.

(Refer Slide Time: 01:10)



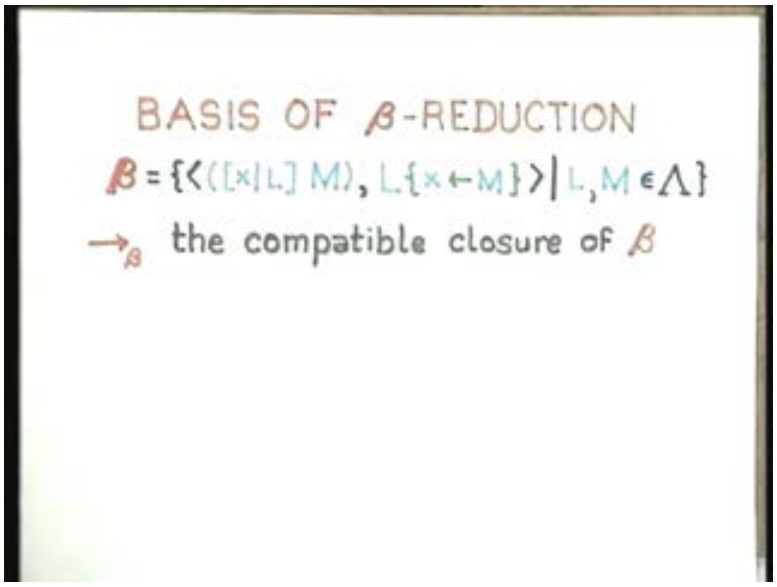
So let us look in some detail at what might be called a model for meaning. So let us go back to the beta reduction and I will define a basis for a reduction as just the first axiom.

What is an axiom?
It is just some enary relation. In the case of reductions an axiom is a binary relation which as a left hand side going to a right hand side. The set of all bold betas is essentially a rewording of the axiom of the first beta axiom for beta reduction. So, for all terms L and M belonging to lambda this ordered pair which means that this application can reduce to this substitution is the basis of the beta reduction.

The normal terminology is to call it a notion of beta reduction. Since we use the word notion in a very general form I will call it a basis of beta reduction but many books will call it a notion of beta reduction. And your one step beta reduction is really what might be called the compatible closure of this set.

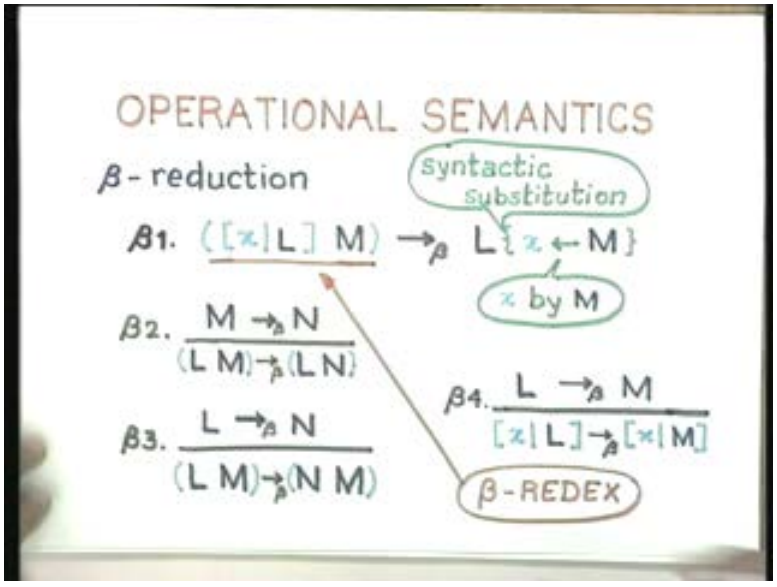
So what is the compatible closure?
It is just is just that when you get these applications in some context then the whole context reduces that way. This is something that we have done before. So the compatible closure really consists of these rules which we gave in the operational semantics.

(Refer Slide Time: 03:40)



So this first beta one is really the basis of the beta reduction and the syntax of the lambda calculus allows only three possible constructs or rather two possible constructs but I can look upon either forms of the application as two possible constructs and this (Refer Slide Time: 4:10) the one step beta reduction that you get by closing it over these three rules is the compatible closure and that is the one step beta reduction.

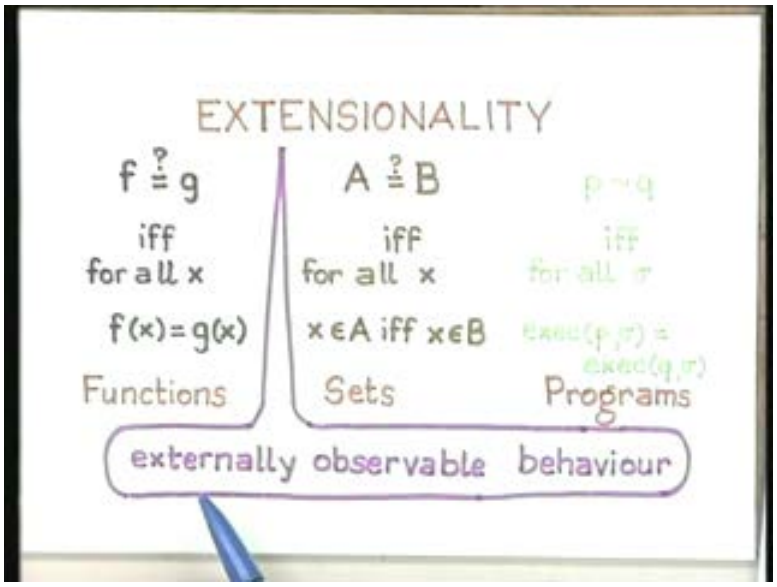
(Refer Slide Time: 04:17)



So this one step beta reduction is just what might be called the compatible closure and the many step beta reduction is the reflexive transitive closure of the one step beta reduction and the equality is just the symmetric transitive closure of the many step beta reduction. So the question of what exactly constitutes equality is something that has been subjected to a lot of philosophical debate. And the main the consensus that is a sort of emerged is when you look at any forms of equality it is important to look at it from an extensional point of view. This extensionality is a is a fairly fundamental philosophical concept which goes over starting from mathematics on to all branches of engineering I mean including electronics for example.

What is extensionality?
Firstly, when two functions are are equal, so two functions f and g are equal. For the purpose of argument let us assume that f and g are both unary functions, it does not matter really if they are enary functions then I will just make this x a vector x. So if f and g are unary functions then f is equal to g if and only if for all values x f(x) gives the same value as g(x). This is what might be called an extensional notion of equality. The effect of the function, if you look at the two functions as black boxes with an input port then for all possible input values they give you the same output.

(Refer Slide Time: 8:37)



Then essentially the two black boxes are indistinguishable and one can be used for the other so that is extensionality. This is as far as functions are concerned. As far as sets are concerned for example it is exactly the same. The basic test that you can perform on a function regarded as a black box is that of supplying inputs and getting outputs.

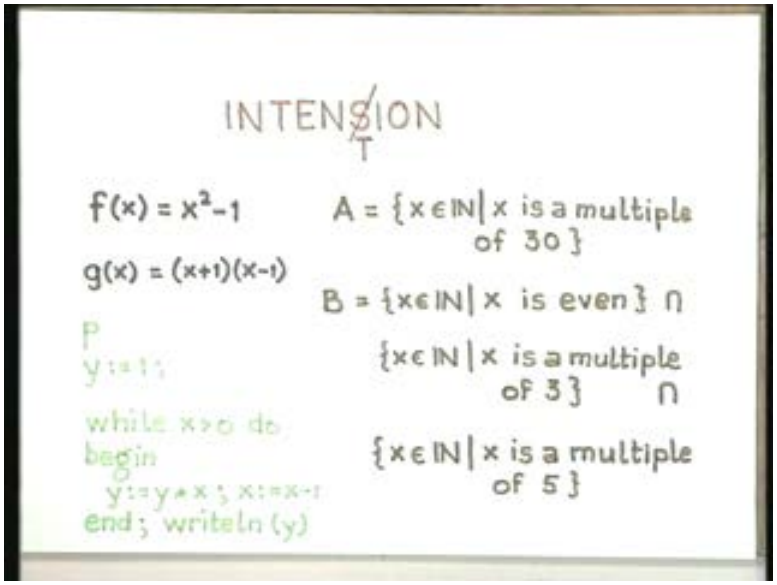
The basic tests you can perform on sets regarded as black boxes is to test from a machine give a input and get a yes or no answer. So, two sets are equal if and only if they contain

the same members. That is like looking at the set as a black box and not looking inside too deeply.

For example, how exactly is the set represented? How exactly is a function representing and so on. Therefore you are taking a black box behavior of the entity in question and assuming that you can perform exhaustive number of tests even if the number of tests you have to perform is accountably large or even uncountable, assuming somehow that you can perform an exhaustive number of tests you decide on that basis when two sets are equal, when two functions are equal and when two programs are equal.

In fact the definition of the semantic equality of two programs is really based on an extensionality principle. Extensionality principle is just that you view an entity as a black box and test it exhaustively. Hence by extensionality what we mean is as opposed to what might be called its intention. So, extensionality is best understood if you understand what intention is and intention is just the construction of the black box, the internal construction of the black box.

(Refer Slide Time: 09:39)



Supposing you remove the cover on the black box and look at its internal details. So you could have this function f of x its internal details are such that it is somehow of the form x square minus 1, you have the g(x) which is of the form x plus 1 (x minus). So the internal details in terms of construction are different for the two. However, since on testing they give the same results you consider these two functions to be equal.

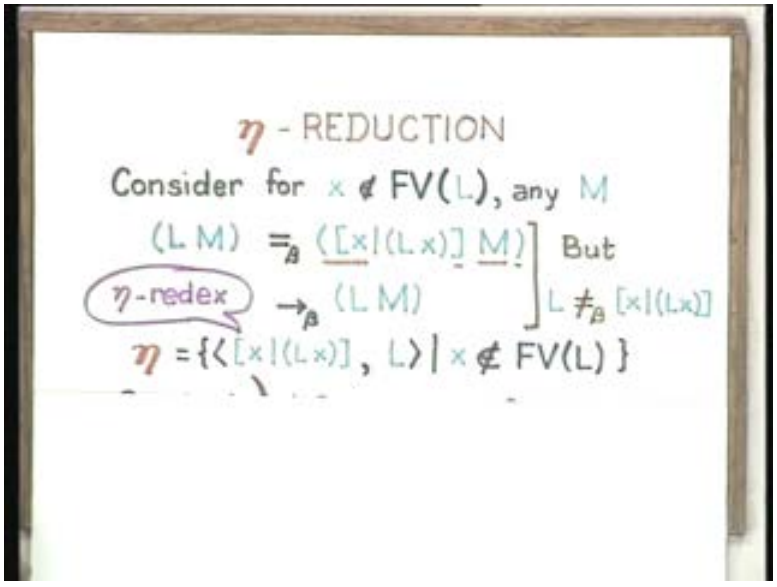
On exhaustive testing for any value of x the black box f and the black box g give you the same results so you consider them to be indistinguishable. And that is in fact what is also true of sets. So if you look at the internal construction of this set it is just that this is a set consisting of all multiples of 30 whereas this is a set where first you construct the set of

all evens then you construct the set of all multiples of 3 then you construct the set of all multiples of 5 and then you take the intersection of all these three states and you get some set. So the internal construction of this set is different from the internal construction of this set the fact that they are both equal and something you can prove through number theory, set theory and so on and so forth but the essential fact is that there internal constructions are different.

For example, they are like two different electronic circuits which are somehow equivalent. So the intention is that we should not worry too much about. When we are talking about equality we are really worried about what might be called the extensional behavior. The most favorite programs in semantics are these. Their internal constructions are different but they are really the same program.

For the moment let us assume that we are talking about x being a natural number so one computes y as a factorial of x by counting downwards and the other computes y as factorial of x by counting upwards. The internal construction of the two programs are different but their behavior in terms of the actual output that you get which is y is the same for all natural numbers x. Hence this is the notion of equality which provides essentially all branches of engineering mathematics and so on.

(Refer Slide Time: 16:25)



When we are talking about equality one of the things that happens with beta reductions is the following: Consider a term L which does not have x as a free variable and consider any term M and now I can look at this term L applied to M and I can look at this term which is L with a lambda abstraction and this lambda abstraction is applied to M. So this lambda abstraction when applied to M means that all free occurrences of x in this will get replaced by M and since and L does not have any free occurrences of X what it means is you will just get L applied to M in a single step of the beta reduction.

In a single step of the beta reduction these two are actually equal by the beta equality principle. But they equal for more important reasons, they equal for extensional reasons. Regardless of whatever might be the argument both these black boxes L and lambda x Lx would give the same results for all possible arguments M. That is the important thing. They would both give the same results for all possible values of M but within beta equality without applying on M we cannot prove that L is beta equal to lambda x Lx. So we infer the equality from its extensional behavior.

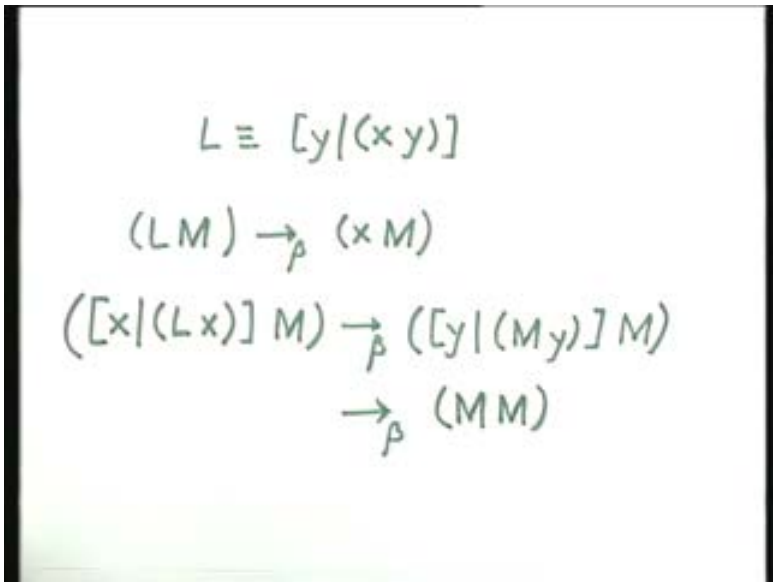
Typically a lambda term is mind to denote a function and if two functions for all possible arguments give the same values then we have considered them equal. But however our beta equality is actually is not strong enough to capture that form of equality in isolation.

So we can only prove that L applied to any argument is beta equal to lambda x Lx applied to the same argument but we cannot prove that L itself is equal to beta equal to lambda x Lx though of course it makes a lot sense to equalize the two. So the beta equality is not powerful enough to take extensionality into account. So what we normally do is if we are looking at lambda calculus as a way of representing functions is that we add an extra notion of reduction or an extra reduction basis and that reduction basis is the eta rule. It is important to realize one thing that L applied to M is beta equal to lambda x Lx applied to M only if x is not a free variable of L.

If x is a free variable of L then these two are not going to be equal because the free occurrences of x within L are all going to be replaced by M and you will get something different. For example, if x is free in L and you apply L to M then M is going to replace some other bound variable and it is not going to replace x. For example, if L were defined as lambda Y x applied to Y where x is a free variable of L the L applied to M is going to beta reduce to x applied to M. However, lambda x Lx applied to M is going to reduce to [lambda(y) (M,y)] the whole thing applied to M again.

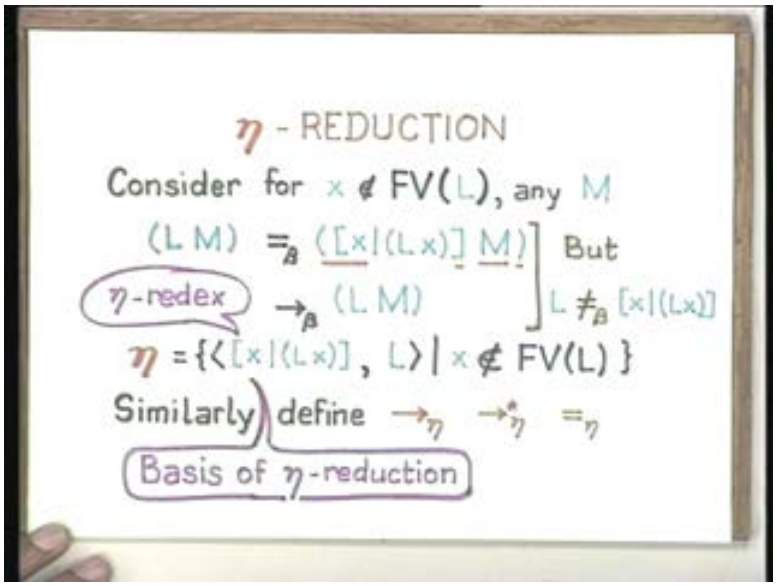
Here all free occurrences of x in this term are going to be replaced by M so what is going to be happen is that x is going to replaced throughout. Even though this reduces further to (MM) M applied to M but these two are not equal this x applied M and M applied M are not equal.

(Refer Slide Time: 19:18)



So it is important to realize that only whenever x is not a free variable of L and you have a term of this form then the effect of this term on all arguments M is going to be the same as the effect of this term L on all values of M. So we will define this notion of reduction or rather this basis of reduction eta and we will define its work a one step eta reduction as the compatible closure of eta over all lambda terms, the many step eta reduction and eta equality as we did for beta equality.

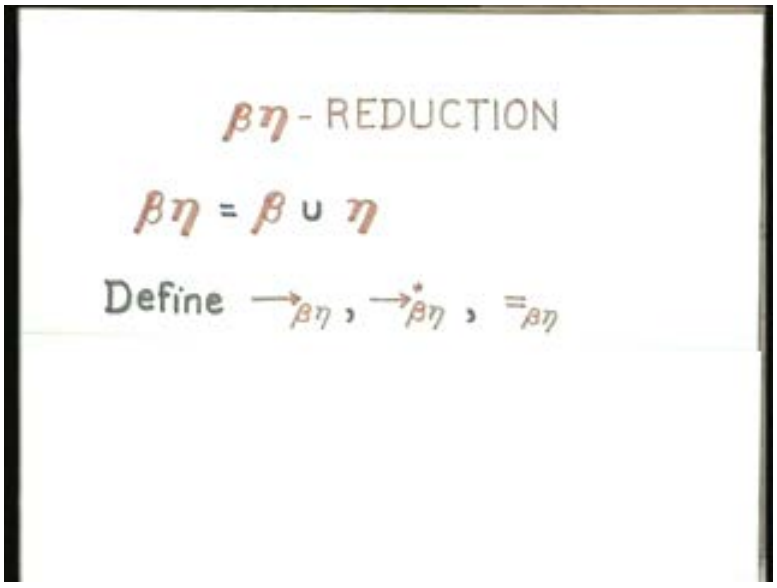
(Refer Slide Time: 20:09)



Now supposing we wanted the extensionality principle actually to be used throughout then what we could do is instead of talking of just beta reduction and eta reduction in isolation we could combine them both. After all beta reduction is fundamental to us as function application so you could combine the two. So you could take the two bases and take the union and then correspondingly give a compatible closure and you will get a one step eta reduction which means you either choose to do a beta reduction or an eta reduction.

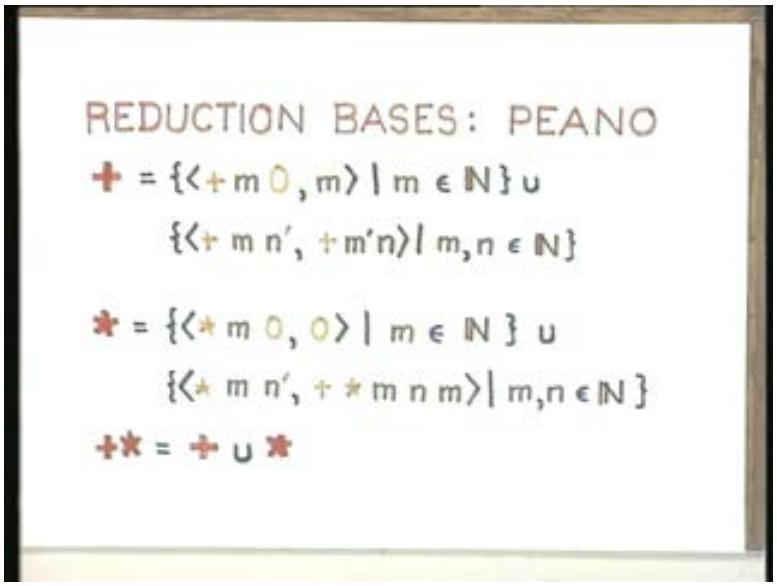
You can intersperse beta reductions and eta reductions in any way you like and you do not have to follow any strict sequentiality. We can define the many step beta eta reduction as the reflexive, transitive closure of the one step beta eta reduction and we could define the beta eta equality. The beta eta equality is really what we would think of as an equality which is compatible with our notion of extensionality.

(Refer Slide Time: 21:30)



You have both function application and extensional equality. So we will not look at, we will not belabor this point; it is just important for us to know that there are various kinds of reductions we might define. And in fact this is not something new because the Peano Arithmetic though I did not actually mention it we actually had two different basis for reduction and we had this language of expressions of Peano Arithmetic so the basis of reductions in Peano Arithmetic were really these two sets and what we actually used in the example of Peano reduction which I did not mention is that we used first the union of these two.

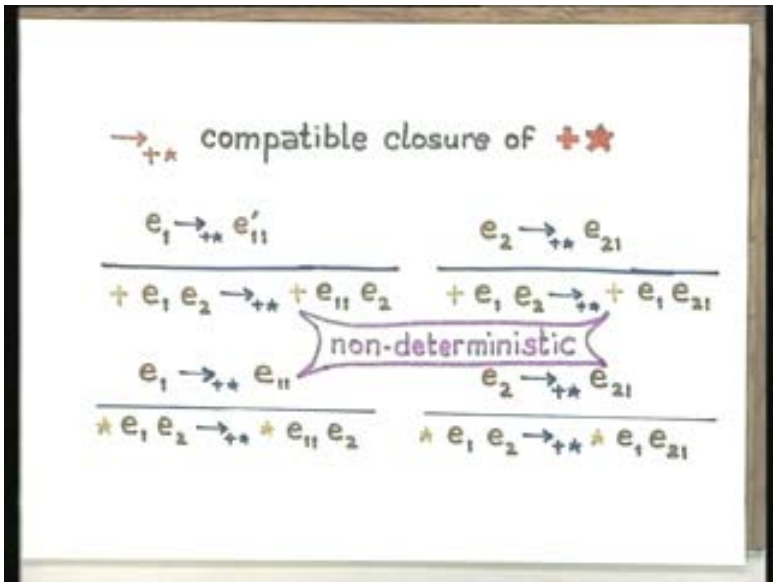
(Refer Slide Time: 22:38)



These are just the two definitions of addition and multiplication represented as binary relations.

The M plus 0 is essentially M for all M belonging to N and M plus successor of N is the successor of M plus N and M multiplied by 0 is equal to 0, M multiplied by the successor of N is the sum of product of M and N and M. So what I had given is rules for Peano Arithmetic and I did not actually specify but we were essentially using this without actually mentioning it and also we were actually using a compatible closure of this which is a one step plus star reduction. Remember that, we omitted these, if you gave these rules it is actually non-deterministic.

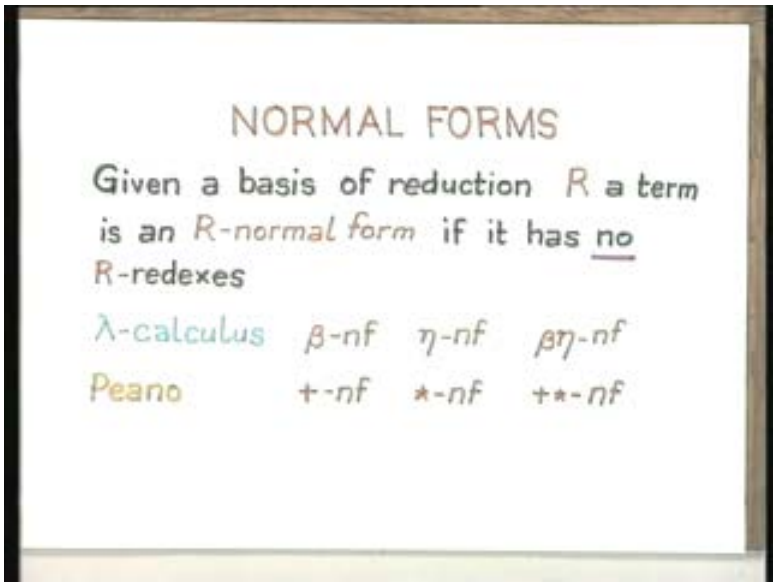
(Refer Slide Time: 23:25)



There could be more than one possible reduction and then actually assuming all these implicitly we actually did this derivation without actually using the rules. We took one possible execution sequence. But essentially in any form of computation including Peano Arithmetic we have first of all non-deterministic sets of rules, secondly we have some notion of combining reduction basis so that you can apply things in arbitrary order and we have the concept of a final value which is what is called a normal form. So what we will do is we will look at normal forms as a general notion.

Remember what we said about meaning? The meaning of a term is the value it reduces to. So, given any kind of reduction basis a term in that language is an R normal form if it has no R redexes. I mean we can define plus redexes, star redexes and we can define for example in the lambda calculus beta redexes, eta redexes, beta eta redexes and so on.

(Refer Slide Time: 25:29)



So, if we look at the original meaning of notion as the value that a term reduces to in the Peano Arithmetic it is really the value represented in the form of these grammar numerals so it is really every number represented as either a 0 or as a successor of a number. That is the representation of a value in that language. In the lambda calculus too we can talk about beta normal forms essentially as a term which has no beta redexes, which is the value of any term. So you perform all the beta reductions still you can find no more beta reductions no more beta redexes and that is the value.

Similarly, you can talk about eta redexes and eta normal forms as a term in an eta normal form if it has no more eta redexes and a beta eta normal form is one where you can neither find a beta redex nor can find an eta redex. And essentially the computation terminates there and that itself is regarded as a value of the original term you started out with. So, when we look at normal forms the important question in the lambda calculus is, does every term have a normal form?

Let us just look at beta reduction. In the Peano Arithmetic it turns out that every term does have a plus star normal form because all arithmetic expressions not having variables will reduce to a value. If you allow variables then you should allow variables also in the basis language of expression and then what will happen is you will get a normal form which is either a value represented as 0 or the successor of some number or you will find it as a sequence of successors of some variable or a variable itself if you allow variables into Peano Arithmetic and into the expression language of Peano Arithmetic.

In the case of Peano Arithmetic you actually have all terminating computations. You always have a plus star normal form. But in the case of lambda calculus we had this horrible term called omega for example which does not have a normal form because at every step of this computation you can always find a beta redex. The plus zero dash X

would reduce to, if you go through the beta reduction rules it should finally reduce to plus x prime 0 and therefore it will also reduce to x prime so that would be the normal form.

If you follow the rules systematically it will reduce finally to x prime. Our rules for plus is asymmetric. What it means is you will gradually be getting the successor from the right hand side argument to the left hand side argument. So you should finally get a zero on the right hand side. Or if there are two variables then you will get a plus expression in terms of the two variables.

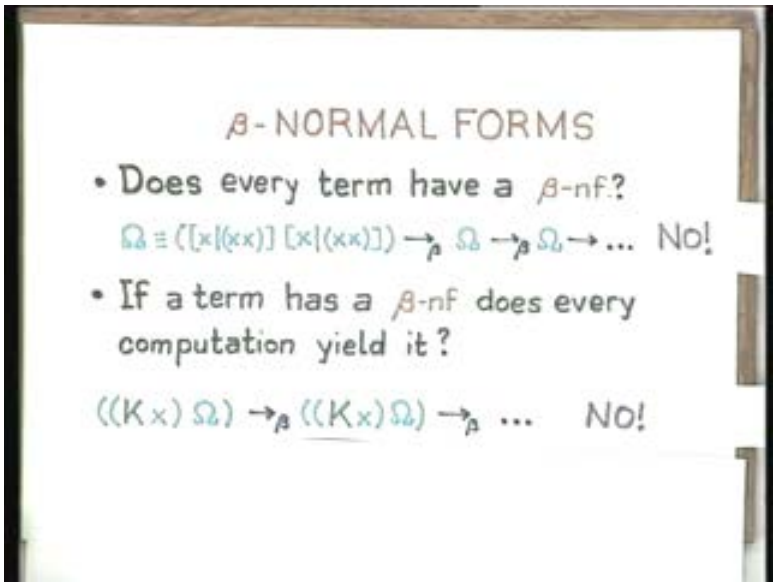
So, for example plus of x,y prime would eventually just give you plus of x prime and y, that would be your normal form. But you do have normal forms always. You have a step where you really you cannot do any more reductions. Whereas in the case of the pure lambda calculus there are terms which always have beta redexes regardless of how many ever reductions you perform and so therefore not every term has a normal form.

So the next question is, we have non deterministic computations, so now supposing instead of pursuing one form of computation I pursue some other form of computation, I have a collection of beta eta redexes and I have to choose one, I can choose one instead of the other. So I have several possible computational sequences, several beta reductions possible.

Supposing you have a term which has at least one beta normal form and it starts off with several possible beta redexes so that means you have several computational sequences do all those computation sequences yield the same normal form.

Again if you look at Peano Arithmetic that is in fact the very basis of our notion of computation you will find that all computations that you might do finally yield the same normal form. We have grown up with it and we have taken it for granted. But it is an important question; if the lambda calculus is really trying to give you a fundamental picture of computation then does every computation of a term which has a beta normal form give you the same beta normal form. For this again the answer is no.

(Refer Slide Time: 32:01)



So, if you take this term K the combinator K is the projection combinator. It is this combinatory. It takes two arguments one by one and it gives you the first argument. However, because of the non-deterministic nature of beta reduction there is absolutely no reason why I should apply K to x and omega and get back this normal form x, x is a normal form I mean there are no beta redexes.

On the other hand, this non-determinism allows me to do a reduction on the omega as far as I want. And at any point I might decide to do the reduction of Kx omega. So there are a whole lot of finite terminating computations which all terminate in the normal form x but there is also an infinite non terminating computation where I have chosen never to apply K the beta redex Kx omega but always apply the beta redex omega itself. Since omega itself is an application term if I always choose to apply the beta redex omega then I get a non-terminating computation.

Now what has happened is there are some fundamental notions in which the lambda calculus is actually different from our normal understanding of computations in mathematics and so on and so forth. So now you have non-determinism, you had non determinism also in the Peano Arithmetic there was no problem.

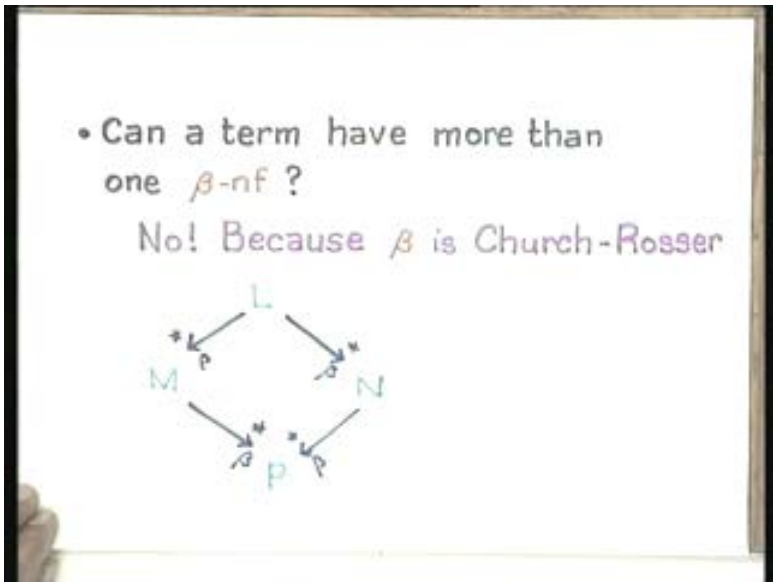
The first thing was that the Peano Arithmetic always guaranteed a normal form which the lambda calculus does not. The next thing is, even though there was non-determinism the Peano Arithmetic actually guarantees that every possible computation yields the same normal form which the lambda calculus does not. And lastly supposing if the lambda calculus does not guarantee these things then does it at least guarantee that, where two different computational sequences exists which both yield normal forms is it possible that a term can have more than one beta normal form.

In fact if you look at the Peano Arithmetic, since all computations terminate and even though there is non-determinism they always terminate in the same normal form so you cannot have two distinct normal forms. Now can a term have more than one beta normal form?
So the fundamental question we are asking is can a term have more than one meaning depending upon how you execute the term.

So, if you think of the notion of meaning as an intrinsic meaning then what we would like in order for computation to be possible is that a term should not have more than one meaning. There should be an intrinsic meaning. If it has more than one beta normal form it means that it has two different intrinsic meanings and the meaning is dependent on the order of computations which means that your notion of meaning is no longer a function from syntax to semantics, it is something very peculiarly concerned with the order of execution. This is in fact true of many programming languages, if you take this short circuit Boolean evaluation for Boolean operators like AND and OR then our intrinsic meaning of the Boolean operations AND and OR are that they are both commutative. But if you had an implementation with short circuit evaluation then there is no guarantee at all that you will actually get the same meaning.

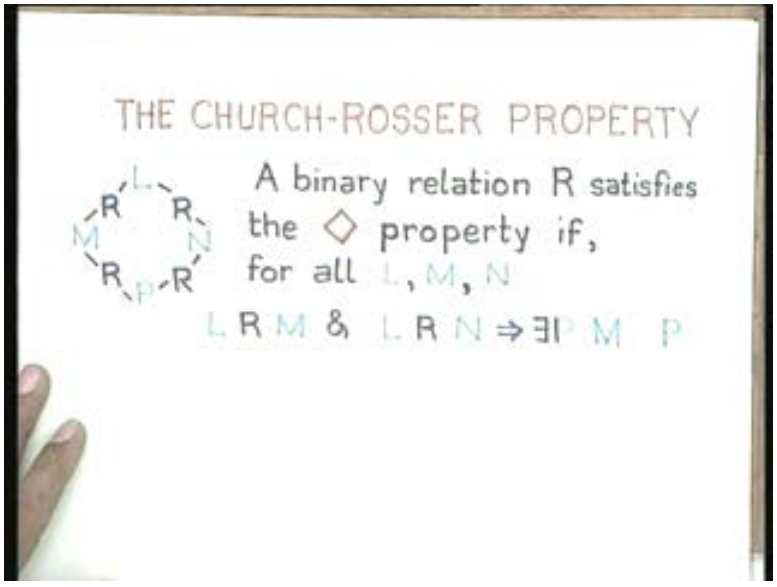
The intrinsic meaning is that they are commutative but in programs we know especially in the presence of side effects that if you flip the two operands of AND you can get different results because the meaning is somehow not intrinsic enough to be determined by this semantics but it is somehow dependent on the computation sequence. So what is the fundamental property which will guarantee that a given term has only one intrinsic meaning and that is the Church-Rosser property. So this is finally the only thing in which the lambda calculus actually does something that we expect. So it turns out to your first perception that does every term have a normal form, well that is destroyed. If a term has a normal form does every computation yield the same normal form? It is not clear.

(Refer Slide Time: 38:45)



But if the condition is that if your system of reduction is Church-Rosser that means it satisfies this property then it is guaranteed that a term can never have more than one normal form. And in fact that is a fundamental property of all these reduction systems that we are talking about that they should be Church-Rosser otherwise meaning is no longer a function of the syntax meaning is then very highly dependent on execution.

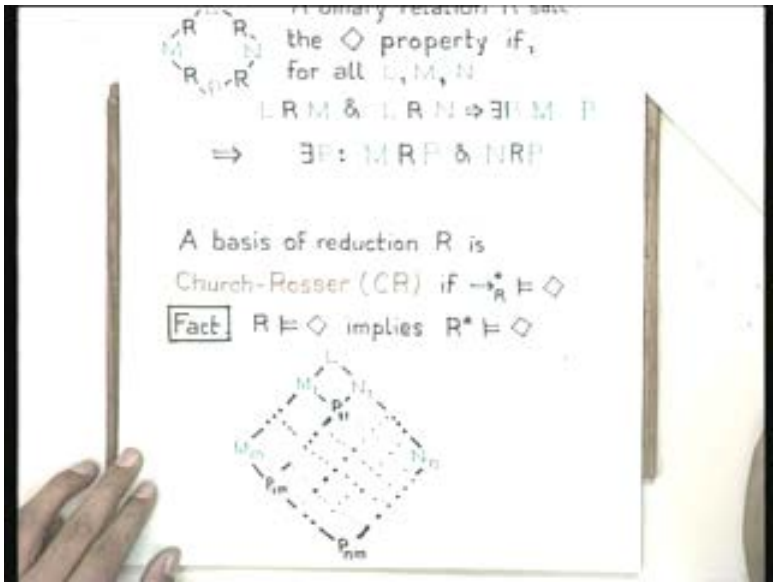
(Refer Slide Time: 40:18)



Actually it turns out that eta reduction is also a Church-Rosser. Let us look at the Church-Rosser property in some detail. The proof here is really the fundamental property. Now we can talk about the Church-Rosser property for any kind of reduction basis. So I will say that a binary relation R this is not necessarily a reduction relation this is any arbitrary binary relation on terms of the language.

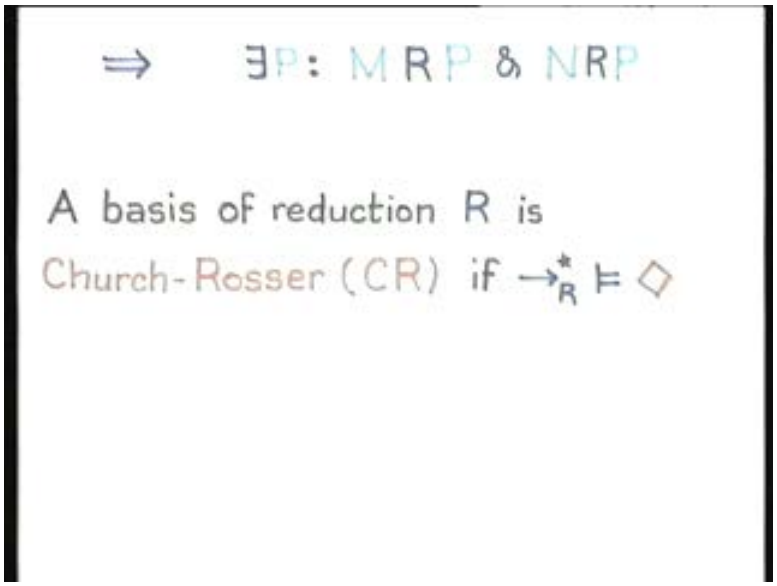
A binary relation R satisfies the diamond property, if this diamond is satisfied. That means given that for all L, M and N if it is true that whenever LM belongs to this relation and LN belongs to this relation if it is guaranteed that you can find a P such that M and P are in this relation and N and P are also in this relation then you would say that this relation satisfies the diamond property. In the case of notions of reduction what we are really looking at is this, the basis of reduction.

(Refer Slide Time: 41:28)



Now supposing instead of just being an arbitrary binary relation if your binary relation was actually a basis of reduction, now this basis of reduction is Church-Rosser if, from the basis of reduction you get the compatible closure which is the one step reduction, from that one step reduction you do a reflexive transitive closure you get a many step reduction and if this many step reduction satisfies the diamond property then you would say that the original basis of reduction is Church-Rosser or satisfies the Church-Rosser property. And this is in fact the fundamental thing you have to prove about any functional programming system that you design.

(Refer Slide Time: 42:09)



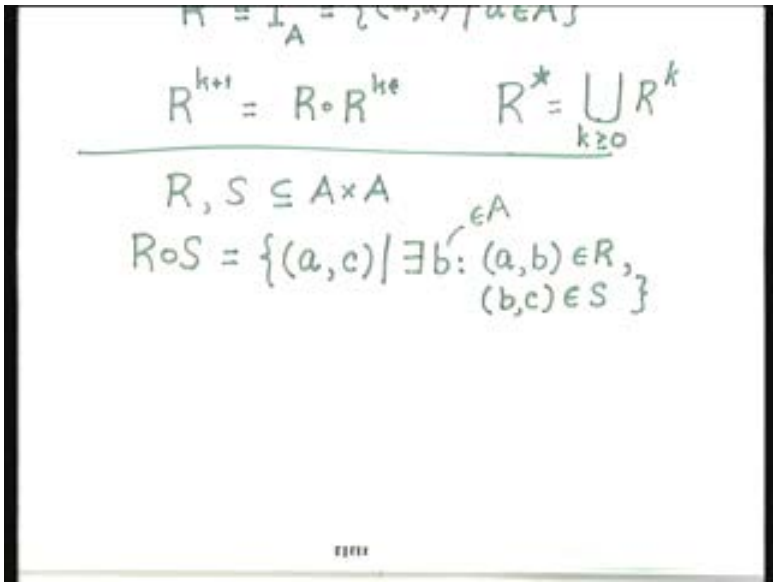
Supposing you come out with a whole set of new operators which have got nothing to do with whatever is there in existing function in programming languages and you claim that you got a functional programming system it is of no use to man or beast unless it satisfies the Church-Rosser property. What it means is that this diamond property should be satisfied by the many step reduction in that functional paradigm. Here is a theorem which has a simple inductive proof.

If a basis of reduction R satisfies the diamond property then its reflexive transitive closure R star also satisfies the diamond property. Actually this is not necessarily a basis of reduction. Any binary relation R satisfies the diamond property then its reflexive transitive closure also satisfies diamond property. The reflexive transitive closure of the relations is: In this case it is not like construction of, we are talking of relation and their reflexive transitive closure so that is not the same as the A power star that we constructed for arbitrary sets, here the R power star is like this. So you have a binary relation on a set. So R is a subset of a set A cross A this is a binary relation so it has various kinds of order pairs XY and so on and so forth. So now I can define R0 to be the identity relation on A.

What is the identity relation on A?
This is to ensure reflexivity. It is just a set of all ordered pairs (a, a) such that a belongs to a. And given R0 I can define R power k plus 1 as being equal to R composed with R power k, this is just a relational composition.

What is relational composition?
If R and S are both binary relations then R composed with S is the set of all ordered pairs (a, c) such that there exists a (b) satisfying the condition (a, b) belongs to R and (b, c) belongs to S. there exists (b) in this set b belongs to A. Now your R power star is just the union of R power k for all k greater than or equal to 0.

(Refer Slide Time: 46:37)

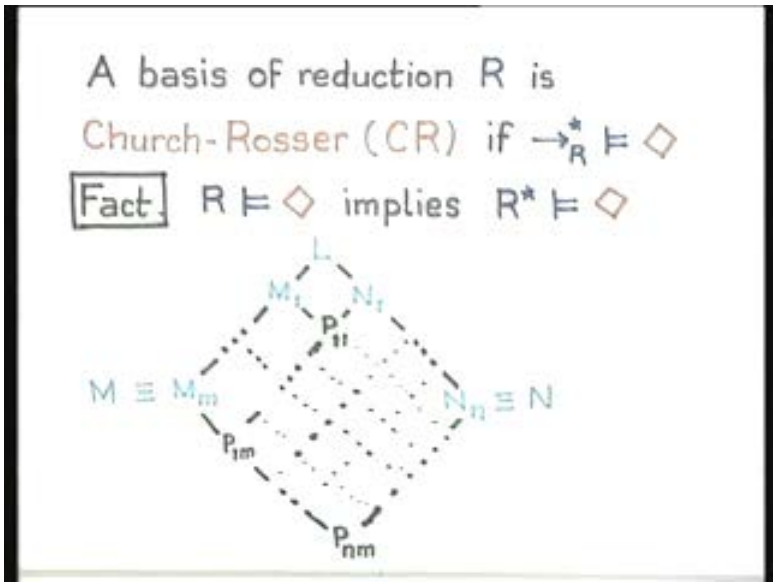


So how does the many step beta reduction go?
It is just you start with a reflexive closure which is a one step beta reduction raised to the power 0 which gives you the same terms it is reflexivity condition and then you compose it successively for k steps and having obtained k steps you get k plus 1 steps by this composition. This is just the relational composition. So the R power star is a finitery relational composition. That means you do not allow infinitery compositions.

If you allow only a finitery composition and if R already satisfies the diamond property then R star also satisfies the diamond property and that proof is very easy. Supposing you had L you have to show that if L goes to some M and it goes to some N in two different ways if L is R power star related to M and L is R power star related to N then there exists some P which completes the diamond.

And if L is R power star related to M then what does it mean?
It means that there exists a finite sequence of compositions, let us say L is R star related to M implies that there exists some M such that the ordered pair LM is in R power m which means that I go through M compositions LM_1 is in R, M_1 M_2 is in R, M_2 M_3 is in R and so on and so forth and when I do the compositions I can find this kind of a path. So what it means is LM_2 is in R power 2, LM_3 is in R power 3 LM_4 is in R power 4 and LM is in R power M and so on.

(Refer Slide Time: 49:10)



Similarly, LN is in some R raised to N for some small N and if this relation R satisfies this diamond property then what it means is that given LM_1 and N_1 which are all in this relation, this L M_1 is in R and L N_1 is in R I can find a P_{11} such that M_1 P_{11} is in R and N_1 P_{11} is in R. So I can complete this little diamond and using the same reasoning inductively what it means is that I can complete this larger diamond. So having constructed P_{11} I can use M_1 M_2 P_{11} to construct something that completes another small diamond let us call it P_{12} and using M_2 P_{12} and M_3 I can construct another small diamond which gives me P_{12} and so on I can complete this entire slice.

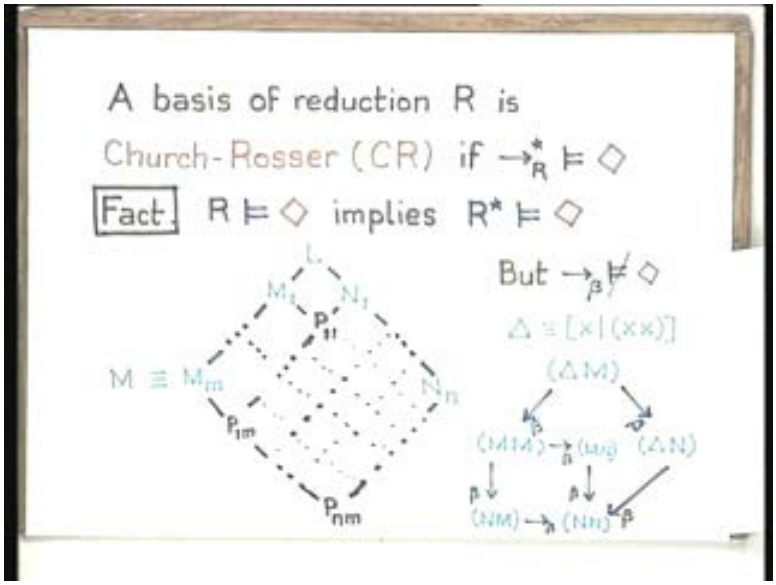
So having completed this slice I can now extend this slice to the next slice. So there is an N step induction inside which there is an M step induction and I can find a P and M. So what is the moral of this story?
Given this fact we can prove that beta reduction is Church-Rosser that the basis of beta reduction is Church-Rosser provided we can prove that one step beta reduction satisfies the diamond property. If we can prove that one step beta reduction satisfies the diamond property then the many step beta reduction also satisfies the diamond property and therefore beta satisfies the diamond property.

However, one step beta reduction does not satisfy the diamond property. Here is the problem, so let me take this delta. You remember that omega the horrible thing which goes on forever, so delta is also a part of that omega. Now you take this delta and you take any term M. Suppose M goes in one step to N the basic assumption here is that we have assumed that M goes on one step beta to N. Then I consider this application delta M and in one step it can go to delta N and then I consider this application delta N which means that all occurrences of x here will be substituted by N by the one step beta reduction rule. By the way there should be a beta here. In one step beta reduction this goes to N applied to N.

However, if instead of choosing the M goes to N reduction if I apply this delta to M directly then what I get is two Ms M applied to M and now I require at least two steps either way to obtain N applied to N. This MM and NN do satisfy the diamond property because you can do this M goes to N beta reduction any time you like but which M are you going to first reduce to N is again non-deterministic.

So in two steps I can go to NN. But I cannot do this in one step. From MM I cannot reduce to NN in a single step and that is where the bottleneck is. But however, beta reduction is still Church-Rosser and so what it means is that such a simple fact cannot be used to prove that beta reduction is Church-Rosser. What it means is that we can use this example somehow.

(Refer Slide Time: 53:59)



We define a new form of beta reduction in which parallel applications of these reductions are possible. That means you can do the reduction across all possible beta redexes that are immediately visible in one shot and consider that one shot reduction which in a single step beta reduction will go over many steps consider that to be a one step parallel beta reduction. Then what you can show is that the star of that is the same as beta reduction star and then you can show that, that parallel beta reduction actually satisfies this diamond property and so beta reduction also satisfies the diamond property.

Then that many step beta reduction therefore also satisfies the diamond property and so it is Church-Rosser. And the important thing about Church Rosser is just that a term can have at most one normal form for any reduction basis R if that reduction basis is Church-Rosser. In different computations you may never get a normal form but if it does have a normal form, if there exists a computation which produces a normal form then all computations which terminate would yield the same normal form provided that basis is

Church-Rosser and that is what the parallel beta reduction does. So I will stop here and go on to the type lambda calculus next time.