Optimization Techniques for Digital VLSI Design

Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Module – 05
Verification
Lecture – 19
LTL/CTL based Verification

Welcome to the module number 5, that is on the verification in the course of optimisation techniques digital VLSI design. So, what we are expecting in this module. So, till its a module number 4 what we have basically seen. So, we have basically seen that, we start with the specification and then we go for some kind of high level specification, formalising in terms of some RTL languages or some hardware definition languages. So, that you can have a specification represented in such a form that it can be implemented in hardware.

Then we go for high level synthesis, then we go for I mean two level or multi level Boolean optimisation, that is to that is gate level synthesis and then we have placement routing and finally, in between we have to also have to plan our test because all the devices to be fabricated are not free of faults as we know already discussed in module number 4 in entirety, that what are the issues of related to test. So, we do lot of test planning also in the flow.

Now, finally what? So, finally, you have a chip which is fabricated tested and sent to the market. Then what is the role verification is playing. So, you might have seen that basically at every stage VLSI design basically is a translation. First we have formed specification, which is written by some human in a as a human being may be in a non technical language, then you convert it into some kind of more formal representation then you have define them in terms of hardware specification languages then you have register transfer level modelling, then you have gate level modelling finally, you lay it out and place and route. But it is actually the same representation of basically what you have thought that, what as I have given at the deign intent that is basically implemented in terms of a layout or in terms of transistors.

Basically so, it has gone through different phases. Now formal verification basically checks mainly that, whether whatever I have intent in my specification is at a very brave very abstract level is very is exactly equivalent to what is being implemented at transistor level. That means, if I wanted to design an adder find that the transistor level implementation should do at add addition or it should implement an adder now how can I do? It very simple is that I apply all possible input cases and see whether at every point at every stage of translation or changing of one way of representation to another, maintains equivalence that is they have to give same outputs for every given input. But now; obviously, they already many many times you are discussing in the course that if your circuit has say at least reasonably complex or at least having more than a 10 inputs everything will blow up, if you are trying to give all possible input cases. And that is actually in feasible way of verifying or even testing anything.

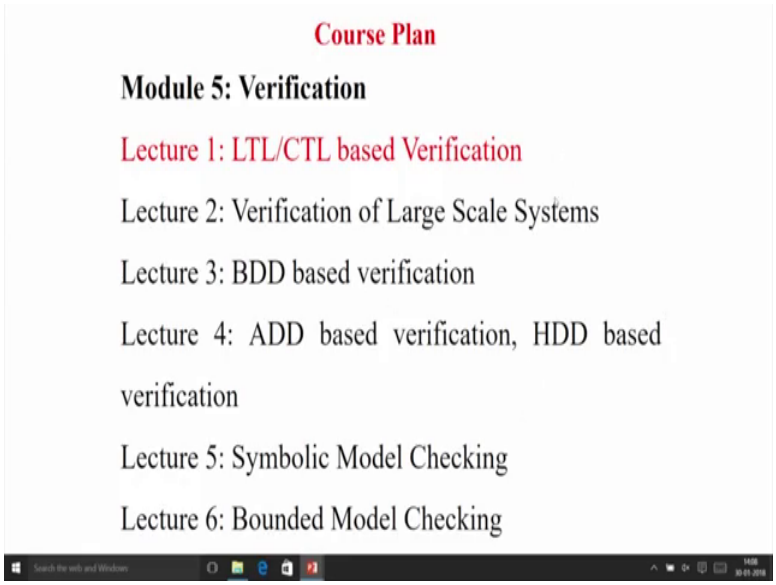
So, in case of verification what we do and we always use the term called formal verification. So, what do you mean by formal verification? There are some other ways also when we call which we call simulating simulation way of verification, but in this course we are restricting to formal verification.

So, what do you mean by formal verification? In this case we are not going to apply test patterns or input or what do I say that patterns or test cases to validate the equivalents rather what we will do? We do it in a formal manner or in the mathematical manner. We will have a model for the system which will be a formal model and then we will try to express our design intake or what we wanted to design in terms of some properties. And then we will do something called model checking in other ways, which will try to verify formally whether the model adheres to this specifications which are in terms of some kind of formula logical formulas. If it is verified then it is mathematically true, that basically your system or your system that is the that is modelled is actually behaving properly with respect to the formulas you want to verify and if there is some problems it will actually give a counter example that is very very important. That if some of the states sequence of the model is not validating or is not satisfying that prop that properties it will give a counter example, which we allow you to find out the what are the bugs and you can recover. So, what will that is the basic idea of formal verification?

So, what we will cover in this module? So, first one will be a very basic because you have to understand because as I told you in this course we have first giving you the

basics of cad for VLSI and then we are building up on it. So, first we will be looking at LTL and CTL based verification that is the core idea that for any kind of formal verification you should first understand what is LTL and CTL model checking. So, which will be covering in this lecture then basically we will going into as the course is on optimisation.

(Refer Slide Time: 04:51)



Then, we will try to see what are the problems with LTL and CTL verification based verification, if you are handling a larger system. Then we will talk about how to verify large scale systems what you have to do, basically that that is one of the code gist idea in which case we will try to see basically how large scale systems can be modelled checked or formally verified in view of that, we will see very important things like something called binary decision diagrams.

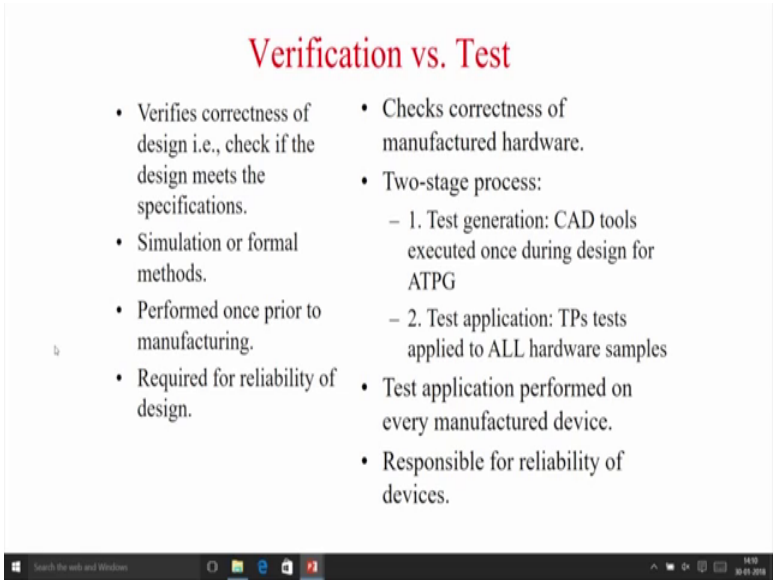
Binary decision diagram is a way of handling large systems by which models large systems by eliminating out your so called the redundancies. So, we will see about binary decision diagram based verification which is also called symbolic model checking. Then binary decision diagram is an abstract is an is actually still it actually minimises the state phase representation by eliminating redundant variable. But in fact, but still it is a bit level, then we will basically see there is something called ADD based verification that is arithmetic addition diagram or high level addition diagram based verification.

That means what? We will not go the bit level we will rather go into higher level of abstraction and try to verify the systems. Like we have already seen in case of test basically when if you are going for the gate level the complexity is high, but when you are going at the RTL level of testing the complexity gates lower down by high level abstract abstraction.

Similar things will also see in the case of verification. Finally, basically as I told you symbolic model checking is nothing, but is a part of when you are talking about large scale verification, symbolic model checking is the way to use to model checking for when you are using binary decision diagram. And finally, what we will see is a something called a bounded model checking, is a bounded model checking is something like we want do not want to check the whole systems in the entirety. May be one set of one path in the state based model can have two million states in that path, but may be you do not want to model set that law, we want to just find out if any bugs exist or whether your property is verifiable with first three steps. So, we can put a bound till what depth or up to what path length we have to search.

So, this steps are basically for your basically trying to use this model checking for large systems, which is the main motto of this course ok. So, let us add this is a preliminary lecture, which we can also find in lot of textbooks or formal verification or many cad for VLSI course covers this lecture, but it will be quite elaborate in that cases, but in our lecture today we will try to give you everything in a nutshell so, that you get a get as a prerequisite.

(Refer Slide Time: 07:10)



So, what is the difference between verification and test, I have already been telling you, but you can look at the slide what it says? Verification verifies correctness of the design. It checks if the design meets the specification or the intent, but test basically means it is basically a device has been fabricated which is already formally verified to be working proper, then only your weight age fabricating whether is the fabrication process any defect has gone.

So, verification means, it is prefabrication testing means its post fabrication. Basically mostly verification are done formally and in test and testing basically is slightly different, we have the device is there you put some test patterns and you verify and you determine whether there is any defect or not. But there are actually testing is a two stage process are the already we have seen, one is test pattern generation or test value that given a circuit what are the inputs you have to give then at what timing you have to give, all this planning's we have seen in the last module is called the test planning.

So, that will give you the test patterns and once it is fabricated you have to apply those test patterns. So, they are two cases basically one is determining the test patterns and one is final application ok. So, testing is a club, but practically speaking code uncode testing is the patterns are there the device is there we physically apply it. But again verification is generally done once prior to manufacturing. That is verification means a design is

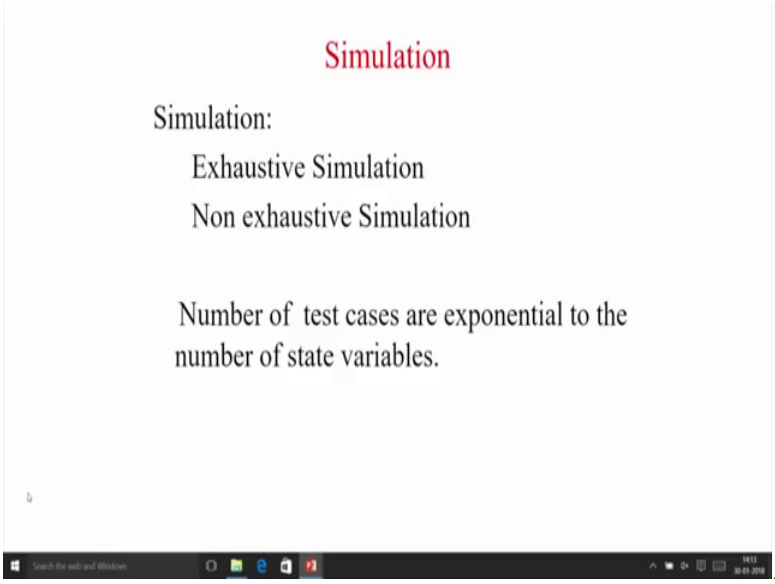
there you have implemented based on terms of this design before implementation in terms of hardware you verify whether its formally adheres to your designing thing.

So, they generally done once, but testing actually as I two steps are there, one is test pattern generation that is again done once because you have determined what are the test patterns to be generated which will be finally, applied, but while applying the test pattern it has to be done for every device. So, basically in a naturally if I have to say what is the device verification and test so, verification actually tries to find out whatever you have designed is formally correct or it is adhering to the design intake.

And test basically is something like its a procedure in which case the device already has been fabricated because it was already prove to be working prove to be formally verified, then the device has been fabricated based on that guarantee, then whether any defect has come in the fabricant stage. That is not at all by any error of the designer something has gone in. So, you want to find out whether that any defect is there or not. So, we apply test pattern and to and we have to do it for all the devices. So, therefore, basically it is called a there is a test so, that is the slight difference between verification and test.

So, as I told you I mean verification can be of many types like what is something called Simulation based verification.

(Refer Slide Time: 09:43)

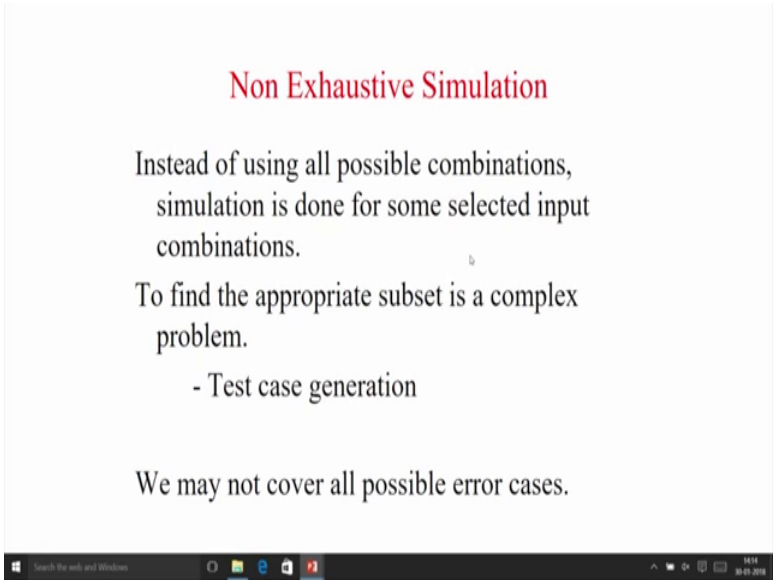


Which we do not do at all means I should not say, but many case is applicable, but generally we are not able to give all kinds of all possible test patterns to verify or all possible test cases I should say, I should not call the test patterns basically in terms of verification, we should basically call it input cases like in the or 101 CS 101 type of courses like C language courses what will happen, basically you write a C code then your invigilator or your teaching assistants come and he or she runs your C code with some test cases.

So, that is some kind of a verification, but in that case the it is only bounded will be verified by the input cases he or she is tried, but it cannot be mathematically proved that is the what you are doing in your labs 101 labs basically that is something called simulation based verification, but it is not possible because it to make it an exhaustive and exhaustive kind of verification is totally out of question. So, it is exponential in the number of test nobody mean tries and also it and.

So, even if you are using a simulation based verification, the size input size is less and also we do not know what is the quality of inputs of the test cases so, there lot of issue. But wherever we are doing a formal verification the idea is that, you are modelling it and you are modelling what are you expecting in terms of formulas and then you are going to model check formally and if you are model checking tells that the formula is valid on this model then you know that the model does not have any bugs with respect to the formulas. So, in that case it is its a guarantee, that your system is going to obey the design intent which was represented in terms of formula.

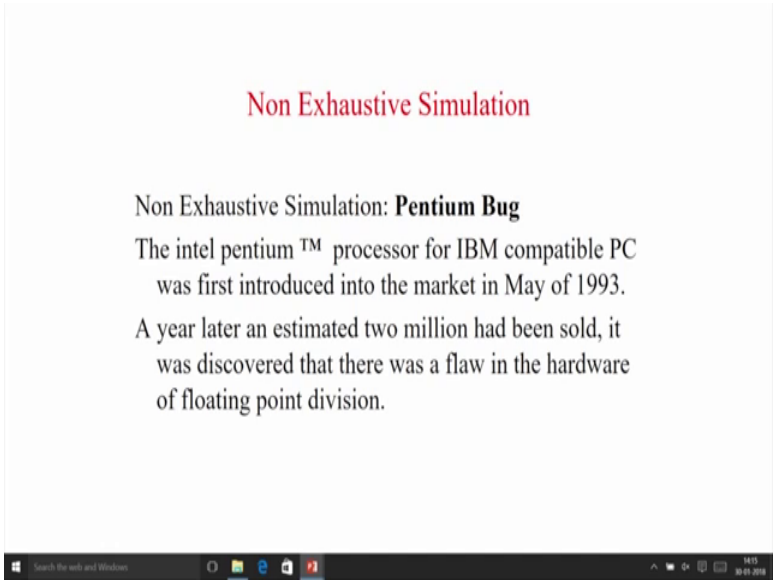
(Refer Slide Time: 11:16)



So, that is why something called non exhaustive simulation. So, non exhaustive simulation means generally some people call non exhaustive simulation based verification, which is mainly used in many of the industries, but in which case they are group of design engineers to design in and there is a group of test engineers or verification engineers, instead of doing it formally basically they design a lot of good test cases and then you find out whether there any bugs there or not.

But in that case test generation is also a lot of lot of humour intervention is required and lot of experience is required as sometimes we may not cover all possible error cases or bugs so, lot of issues.

(Refer Slide Time: 11:49)



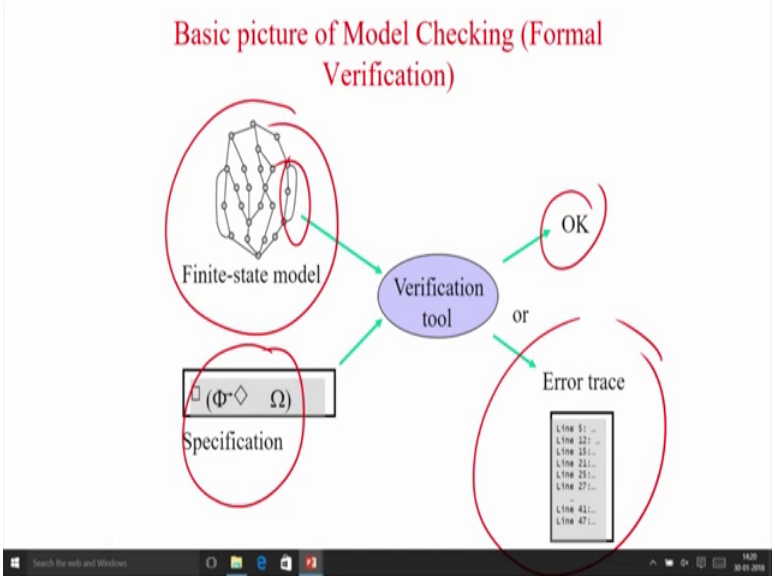
So, in this course and mostly basically people are more formally orient or more oriented towards developing something called formal verification. I think already all we know about pentium bug, there was some and there was some bug in a Apollo space ship, in which case it happened that people have all of them had basically done something called a non formal verification that is simulation based verification or non exhaustive simulation based verification. They were also very intelligent people because designers of pentium designers of Apollo space ship.

So, they are all very very intelligent people, but still they had the stout on some parts on the writing good quality test cases. They might have missed 1 or 2 test cases, which led to the bugs when it the system was diploid in the market or in the system and one led to catastrophic end of the space ship and in case pentium bug the issue was more of finance, because all the chips which had bugs were brought in from the market and they have to be given new chips and all the market reputation came down.

Testing and verification here is a difference again I want to highlight it. When this when they are saying the design is verified to be correct, then basically it assumes that design has the mathematically checked and the testing is not going to again look at technique bug which is coming because of the design engineer. It is looking only for the defects which has been introduced by manufacturing process. So, if you if you design as a problem testing can never find it out. So, therefore, Pentium and the Apollo incident I

have told you, they were mainly because of non exhaustive simulation problem based verification problem.

(Refer Slide Time: 13:09)



So, what people are saying basically there is something called a formal verification method. So, in this case there is a finite state machine model or any model you can use you can use FSM model book automata then also we will see BDT based model as so, many different models can be there.

But is a formal model of your system this has to be done manually and there are lot of issues you can ask me that who does this model, how do they know modelling is proper or not then there is something called specification in which case you write your design intent in terms of some kind of specification formula. Then there is a verification tool which mathematically checks whether this model obeys the specification or not in everything you will get other will get error trace.

Now, I answer your question that, if I make some model mistake in the modelling. Then, but specification is generally very simple as we will quickly see that it is very it is very difficult to met errors there because specification is simple to write your design intent like for example, I say if I request a job to the printer, in sometime I should get a reply. So, this specification will be very simple request imply in future grant. So, this specification writing such smaller compared to the modelling, but modelling a printer

you can understand is a complex task. So, either I if. So, if we so, to get a signal generally this has to be correct as well as the modelling has to be correct.

Modelling means basically this a implementation is a printer physical implementation this has to be translated into a model, whether basically the idea is that people have thought of some kind of model and then they have designed the printer, but generally it may not happen all the time. So, generally people have designed it in a different adhoc manner, I mean I am just coat uncoat saying do not exactly go by the words, but if you say that very specific way of designing system is that you should first have a model, then you have to implement it and then only you can go for the implementation, but generally it does not happen in the practical scenario like that, people generally manufacture or design some systems and generally people go for formal modelling when they have to go for a mode checking.

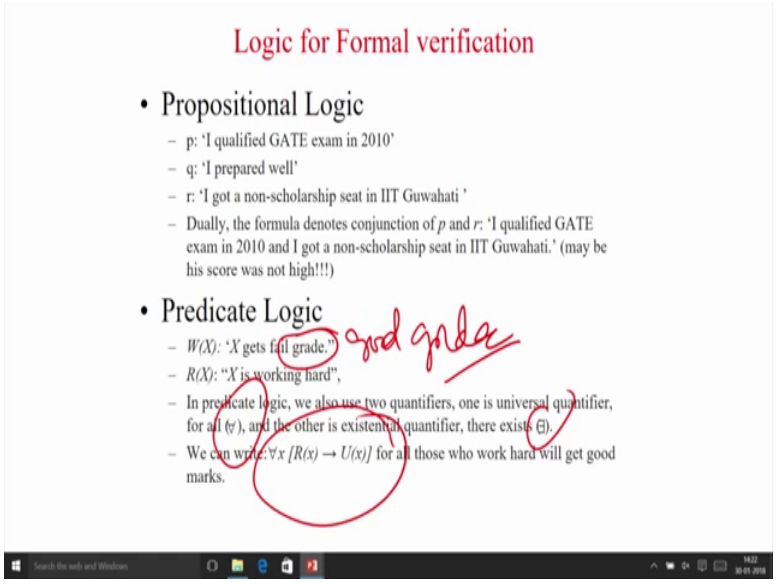
So, now the idea is that even if there is a error in the model, if I may have done some modelling error the system may be proper, but the modelling error may be there, but still you are not going to get an signal you are going to get an error trace. Then you have to find out whether what is the problem there. So, you and find out that this part has a implementation problem then you will quickly say that see your printer implementation this is a implementation problem, then if it the implementation problem very good they will repair it or debug it for the future use, but sometimes they may also say no see our implementation was proper, only the way you have modelling modelled has a problem, you have not captured in a proper modelling you could not capture the modelling in a proper manner.

This is very similar to your course or automata theory sometimes your teacher gives you that there is a string, write a d fa for that. So, main types you make a error in representing in a proper model of a DFA, but then basically in this case what happen then you say this is a bug. So, basically this bug may be as there in the model or the system implemented system, but anyway you will find out what the error is. It is very very rare that even if you there is a you make an error over there still you are going to get an signal its a rare case very very rare case.

So, generally either the system has a bug you will find out or the whether the system does not have a bug, but this finite state machine model representation something might

have come in then you can do. Third picture even if you write a specification wrong then also you may still get a bug, then you can find out that everything is fine this specification was not mentioned properly. So in fact, everywhere going wrong is a very very straight probability. So, therefore, we have a very nice way is expected that you get a fine good modelling permanent model of the system, you write a specification then you automatically give it to the model checker no human intervention, then you find out fine otherwise you are going to get an error trace. So, error trace will tell you where there is a problem. So, in that case you find out what is the problem in the system, that is designed and then you try to rectify we will take lot of practical examples today to see it.

(Refer Slide Time: 16:55)

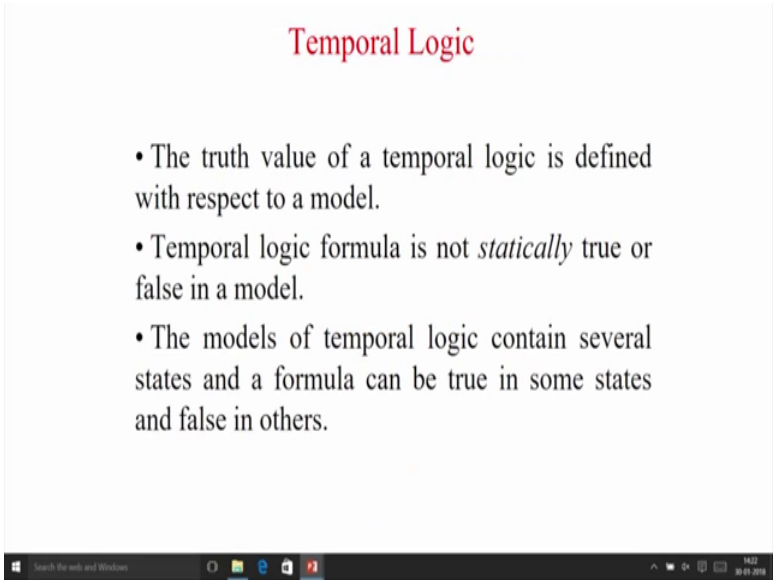


Now, before as we have saying that we will go for formal verification. So, we are just going to quickly cover some logics, which you have already done in your theatrical foundation of computer science. So, what is a propositional logic? So, so if you look at it. So, propositional logic means I have statements like high qualified in gate exam 2010 I prepared well, I got a non scholarship seat in Guwahati. So, if you are making some by heart assertions, this is true this is bad this is good something like it comes under propositional logic what is predicate logic? Predicate logic means you have quantification like W X, X gets a fail grade some variable you have R X, X is working hard.

Now, there is something called for all and they are exist. So, if I write for all for all X, for all X, R x imply U x what does it mean; that means, for all who work hard imply that x is yeah. So, so in fact, basically I should not call it a fail grade it is a reverse way of writing it, I should write it as will x gets good grade ok. So, so basically W X means X grade X gets a good grade. So, in basically it means there is nothing wrong in saying W X, X gets fail grade these are predicate logic, but if I am writing this is sometimes making this means thing non behavioural basically means not practical. So, what did I say for all x R x imply u x means for all those who is working hard will get a fail grade.

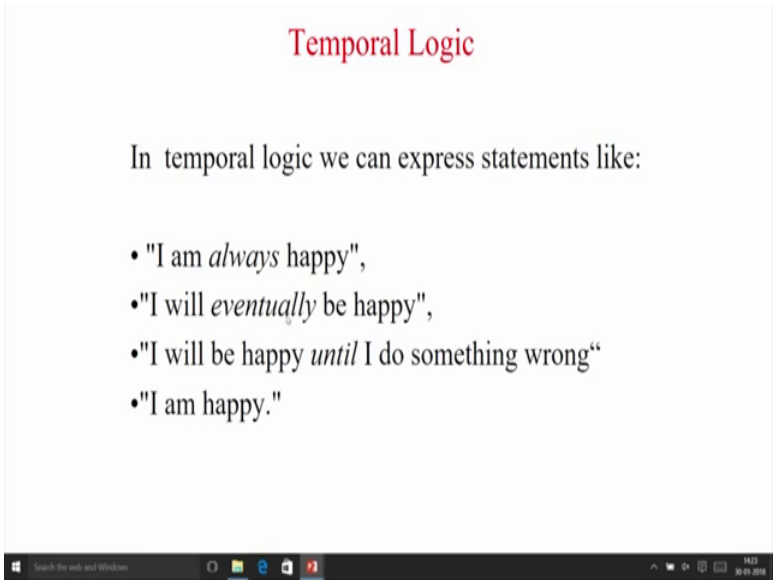
There is no problem in this mathematical represent of predicate logic, but is not a practical way of handling in our real life; that means, this is not a correct when you apply to a real life, because who also are generally will get good grades. So, basically I can write, but mathematically speaking there is no problem. So, predicate logic means that there should there can be a for all and there can be an existential quantifiers. So, it is say that for all R x, there is means for all X R x that is all who is working hard imply u x; that means, we will get a good grade means a fail grade means its a non something very un match other. So, that way you can write, but idea is that propositional logic means there is no quantification it is a direct hit like I get good marks you get bad marks something like that and predicate logic means you can have quantification.

(Refer Slide Time: 18:58)



Now, there is something called temporal logic, which is very very important to this context because we want to find these are all static. Because I prepare whale or those things like I get good marks whoever reads will get a good mark they are all statically true or statically valid or invalid whatever you want to say, but temporal logic means it is not statically true or false in a model, it will if there is number of states in a system in ones it may be true other state it may be false. So, that is actually called temporal logic that is what use temporal.

(Refer Slide Time: 19:27)

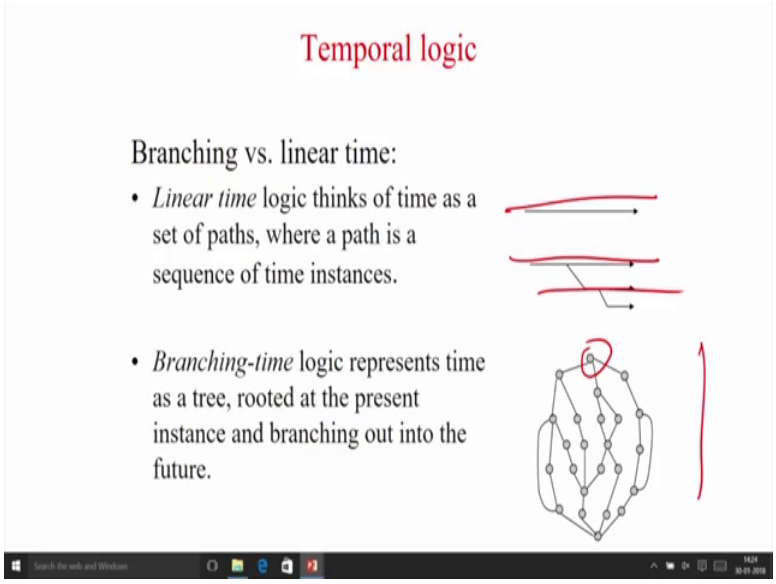


Some examples I am always happy; that means, there is something which is valid in all states, I will be eventually happy; that means, there is some future state, I will be happy I will be happy unreliable something good I am happy. So, I am happy basically you can say I means basically some kind of a propositional logic, but in a temporal logic is a super set you can think because I mean if you say that something is true in all the states. So, that is nothing but your propositional logic.

But temporal logic means it can change over state, but even if it does not change over state it is direct. So, temporal logic you can you can think as a bigger set, but not going into all this subsets superset relations now, but temporal logic you have to understand something which can change over type that is the idea and in the worst case something may not be changing over time it can be valid in all this states or invalid in all this state's, but will also come on the temporal logic.

So, it is saying I am always happy eventually happy until. So, there is something which will be true in some states and true in false in some state. So, that is actually called the temporal property and most of our formulas will be writing today will be on the temporal logic.

(Refer Slide Time: 20:24)

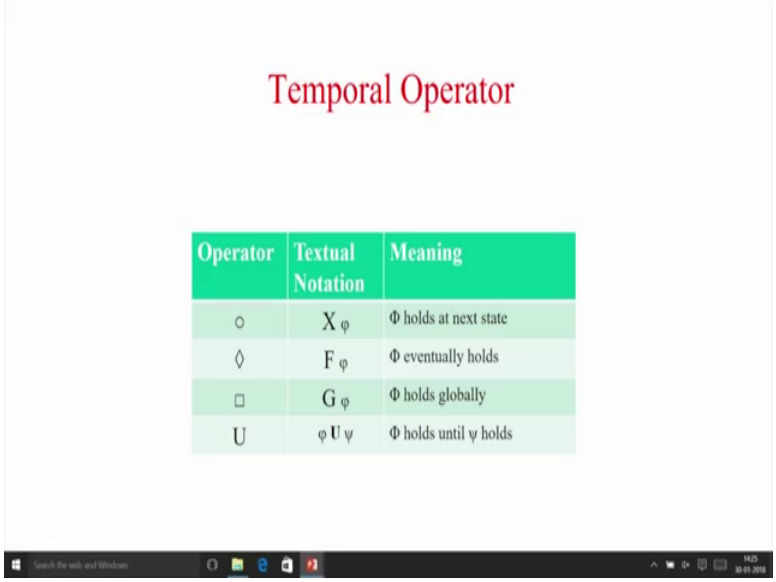


There are two type of very important temporal logic, one is called branching and linear time. So, linear time basically you can think thinks of time as a set of paths, where a path is a sequence of time instances. So, this is linear time that is linear time temporal logic we have might have heard the name always many times. So, basically it is depending on some kind of paths and branching time logic basically represents a tree and there can be different path branching out of it.

So, we will mainly we will try to cover this, but I mean I do not want to go into all these details that which is a super set, which is a subset, basically there is some there is something called LTL and there is some called CTL, which will try to see mainly will try to see CTL properties today and basically which will give because, but you cannot say that one is these super set of other because some there is something which cannot be written in CTL and there can be something which cannot be written in LTL, and all this things are there. But this for that you have to need to know all details about CTL and LTL model checking. So, you can take any standard textbook or cad for VLSI or verification you will be finding it. But what I am trying to give this course basically is an

optimisation part of verification. So, CTL will also require optimisation LTL also will require verification.

(Refer Slide Time: 21:34)



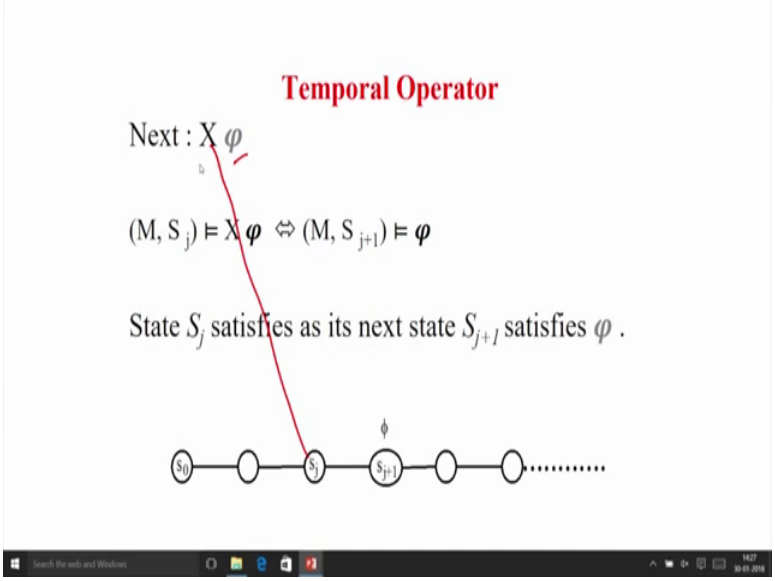
So, in our case basically we will be mainly looking at your CTL based model checking and. So, that I mean basically there all other techniques of optimisation we will be talking about for CTL will also be applicable for LTL or any other model checking you apply.

So, unnecessarily means in this course, if you are going to teach you about LTL, CTL then which is a super set of other and what is possibly in what and what is not possible in this thing, it will actually divert it and it is mainly available in any kind of cad for VLSI course. There is something called CTL star which basically takes care of both LTL and CTL model checking this is super set and it can handle both.

So, I mean those things you can find out in the textbook any kind of a textbook, but ok. So, anyway, but there is some formal definitions I have avenue proposition logic, predicate logic, temporal logic, what is branching time, what is linear time and what is branching time all those definitions basically just I have given you. But mainly will be diving into deep depth in c for CTL model checking and try to find out that what are the issues there in terms of complexity so that we can handle it in future because our course is mainly focusing complexity issues.

So, there are some temporal operators like there exist an X phi X; that means, there is something which hold in the next state. F phi; that means, this formula phi is a temporal formula which eventually in future. Global means globally every state it should hold and phi ample side; that means, there phi will be phi hold until zhi holds I will take all examples and elaborate it carefully and in details, but some temporal operators are next state future globally and until.

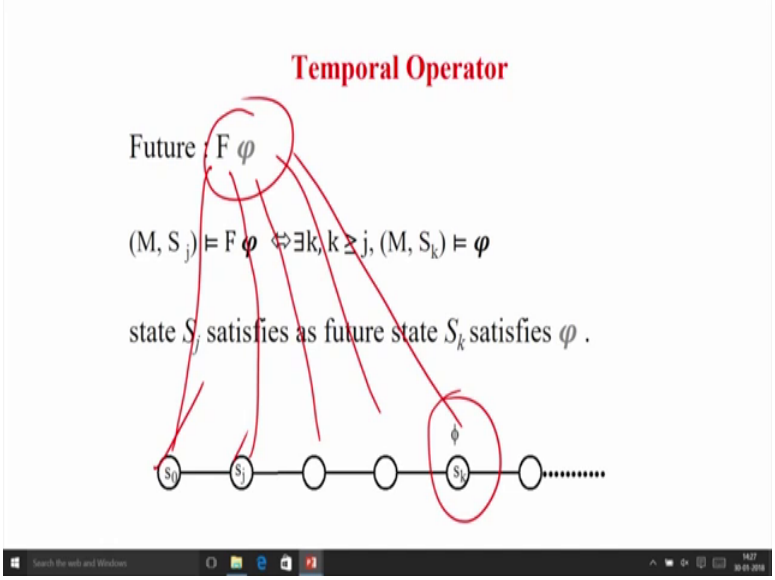
(Refer Slide Time: 23:02)



So, we will take some start taking examples next state fine. So, in this case you can say that in next state phi is 2.

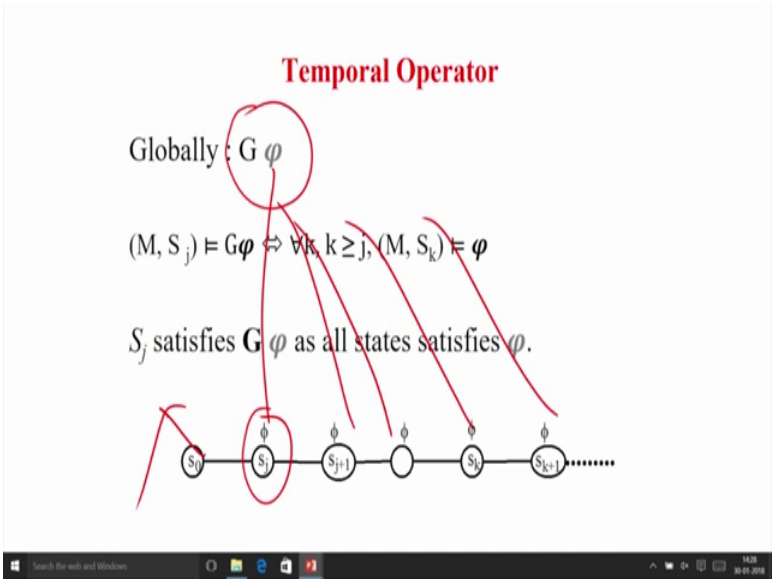
So, you can write in state X a this property is true the temporal property is true that in next state phi is true. So, in this state I can write this one is true. So, this is the S j satisfies at the S satisfies this formula because in next state from S j this one x means next state in future.

(Refer Slide Time: 23:27)



So, here I am writing in future phi is true; that means, this is be holding in this state as well as holding in this state because there exist some state in future where phi is true. So, I can write in state S 0, this formula is true this is true, this is true, this is true, here also I can write this is true because presence state is also future, that one thing you have to write then state S k also I can write in future phi is true because present is also a kind of future globally.

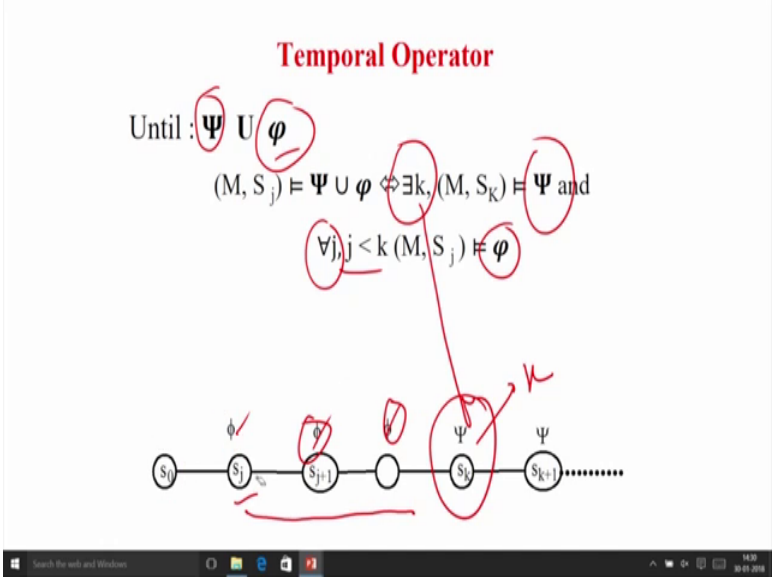
(Refer Slide Time: 23:48)



So, in this globally means some states from some state were globally in all other states from the path starting from that node, because then legally we are going for branching time logic. So, like for example, these state satisfies this all this formulas, because in all the states downwards trace phi is true. So, I can write from all the state globally phi is true because in all states phi is true of course, these S 0 does not is not satisfying this formula.

Because in this state S 0 itself phi is not true. So, globally means from the staring state as well all states is actually having the whatever formula should hold all this thing important one is this one.

(Refer Slide Time: 24:23)



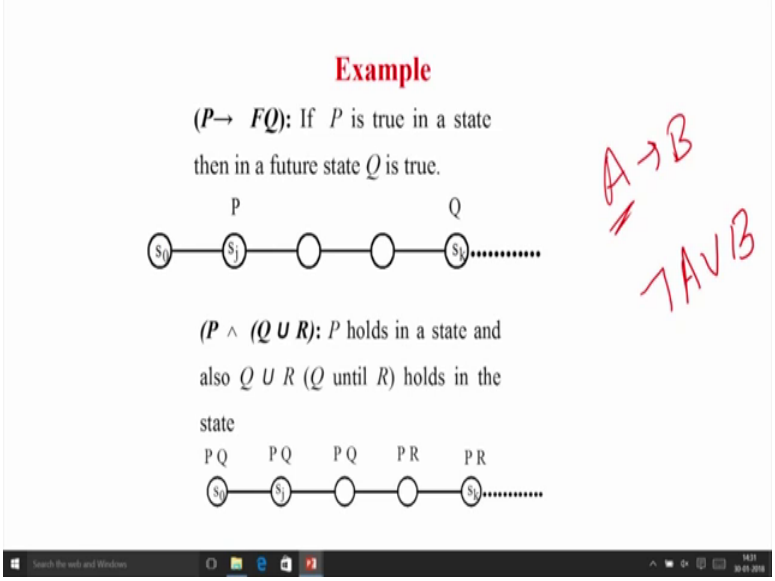
That is basically zhi until phi that until operator that is very important. So, what it says? Zhi sorry zhi should hold until phi holds; that means, phi will keep on holding at a state from where zhi is going to hold; that means, what it says zhi until phi. So, until phi whenever phi is there; that means, something you are very safe, but whenever phi goes off zhi should take it true, that is zhi until phi. So, until phi everywhere will phi is there no problem wherever phi stops zhi should take over.

So, actually this state's satisfies it, but of course, you can say that this state satisfies because from here zhi phi phi phi and from where phi dives down this one takes over. For in this case you can say that is not true because in this not satisfied in S naught,

because here phi is not true still zhi is not taking over. So, therefore, in this state this state is not satisfiable; that means, you can think something like this is a safety property kind.

So, from wherever the safety property or some good property starts holding, then phi need not be there, but still then he has to or this guy has to be the backup kind of a thing. So, just to remember this, but zhi until phi is something represent this the mathematical formula, which says that there exist a k, some k is there where basically zhi is holding. And before that for all j, j less than equal to k for all j less than equal to k your phi should hold. So, basically in S j it will be true because there exist a k where zhi is true and any other case like this, this and this which is less than equal to this basically k your zhi is holding, but S 0 will not satisfy this ok. So, you can say S j S j i plus 1 all satisfy this temporal property.

(Refer Slide Time: 26:05)



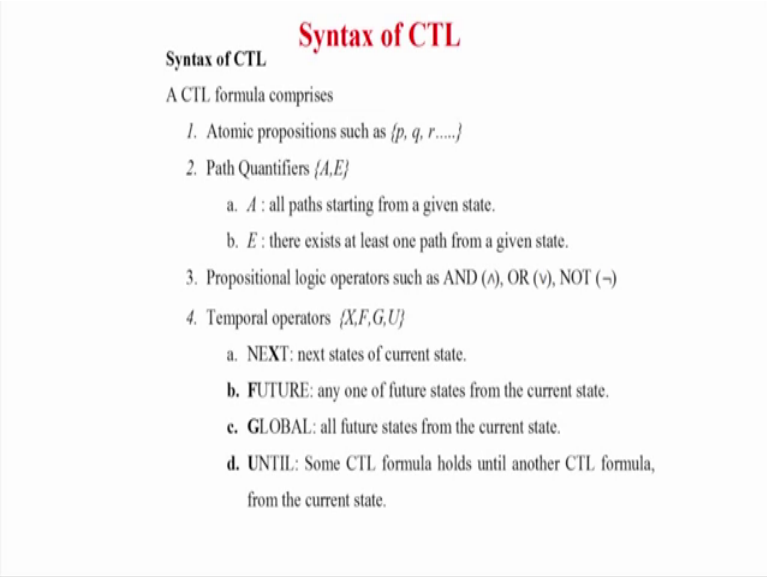
Now, we will take slightly more combinations like in this case it is saying P imply in future Q; that means, what? One many important thing I should tell you might have done this in logic, but you still should remember.

So, one does A imply B means if A is true B has to be true or there is something called that is obvious, but there is something called vacuous truth here. So, if some place A is false in that case B becomes vacuous truth that has to be remembered. So, a imply b means basically not A or B. So, if some place basically A is false, but B is true in that we called a vacuous truthbasically. Means if it implies that if A is true, then this should

happen if some a itself is false they assume it to be vacuously true, but j otherwise if A is true then b has to be true otherwise the property is not holding. So, that is one thing you have to keep in mind. So, what do we say? It is saying P implies in future Q. So, here P is true which will imply that in future Q is going to be true. So, it is true and this state, but if I it was something tell me that whether S 0 what is what is the keys? But in S 0 you can say that P itself is not true, but in future Q is true so; that means, it is vacuously true in this state.

So, that you have just keep a thing in mind that what is mean by vacuous truth. Here it is say next one is saying P imply and Q until R. So, let us see in the state S 0. So, in this state S 0. So, let me just zoom this up for you. So, in this case what we say. So, P and Q until R. So, Q until R means it should be QQQ. So, from this is QQ sorry. So, that is QQQ and Q after that R should start holding and, but in all the states P should be true. So, if you look at it is holding here you can see all the states P is holding and the other part that is end part of it, that is QQ and whenever Q stop R should start holding. So, this state sequence actually satisfy at this property.

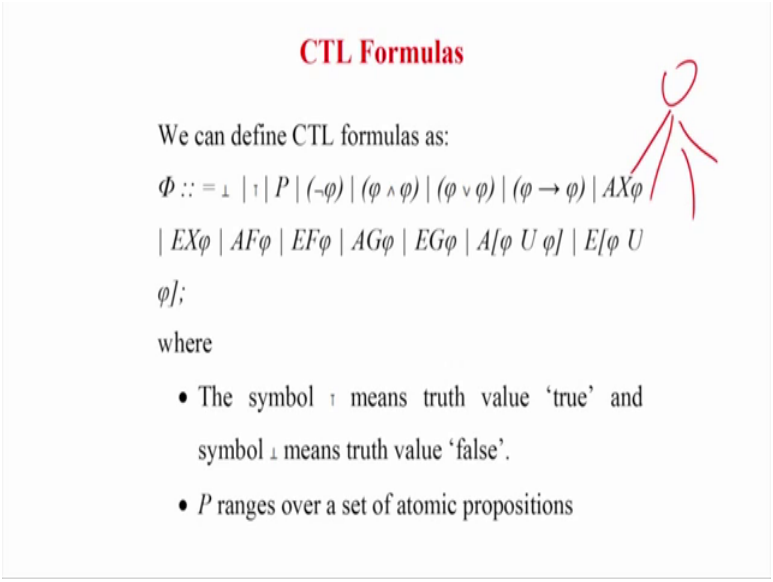
(Refer Slide Time: 28:07)



Now, this is something called CTL computational three logic as I told you apart LTL is also another way of serving the same purpose, but again the power of LTL and CTL model check is slightly different, but in this course we are not trying to go into all those expects, basically we are trying to see CTL and what are the complexity involved here

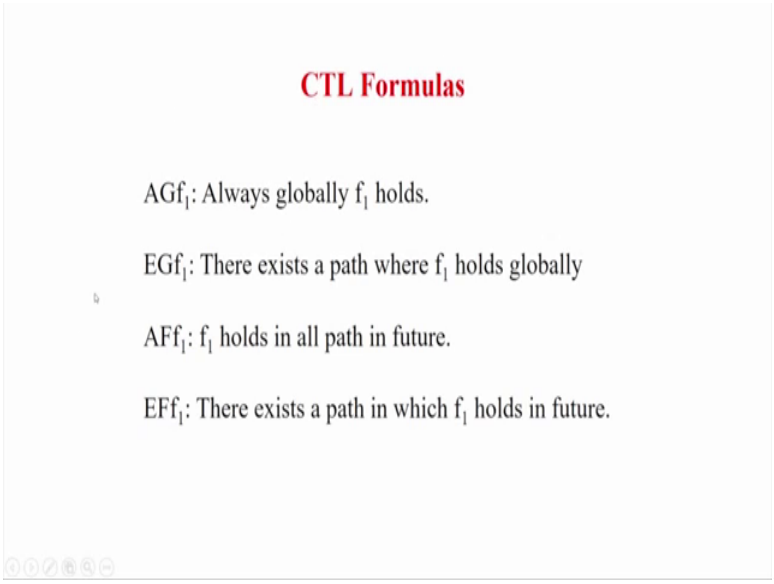
so, that we can try to resolve those complexity issues and go for optimising verification tools for large scale circuits. So, what is syntax of CTL? So, CTL formula that is your formulas which you have to write to design to express your design intent. So, there should be lot of atomic propositions. So, what are atomic propositions? I send a request it is raining, he is a good boy something like that. Then there is some path quantifiers because as I have told you CTL means, there are lot of paths from the route node there can be lot of path branching time. So, you can say that A means over all path and E means there exists a path. So, there can be two ways of representing this all paths or there exists a path because a branching time logic. Then propositional logic operators like and or not will; obviously, be there and temporal operators we have next state future globally until. So, basically these are the syntax of CTL.

(Refer Slide Time: 29:19)



Then, there are some CTL formulas like for example, true false and or not imply everything will be there like in this case it say there exists a path where globally in all path globally. So, we can have path quantification over here because from a tree we will have lot of nodes. So, from the root we will have lot of path. So, you have to select any or two of them right.

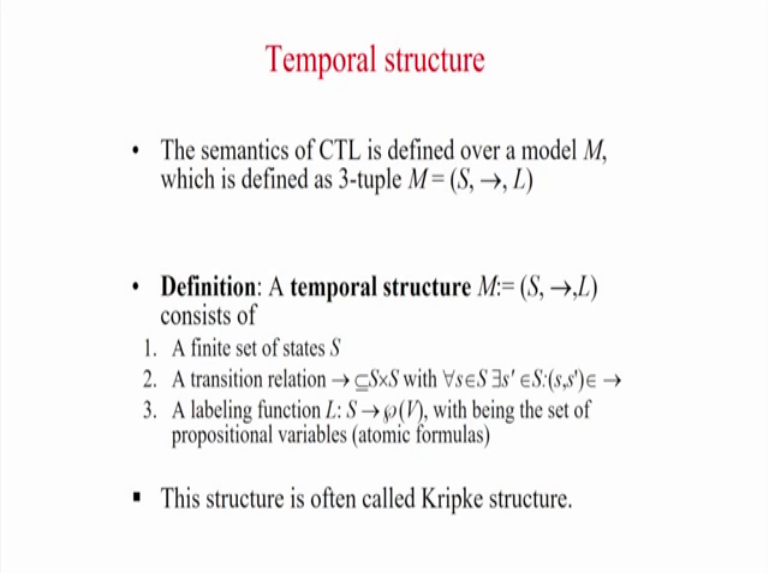
(Refer Slide Time: 29:41)



For some examples are given over here. So, it is saying AF AG f 1 means always globally that is in all path globally f one holds.

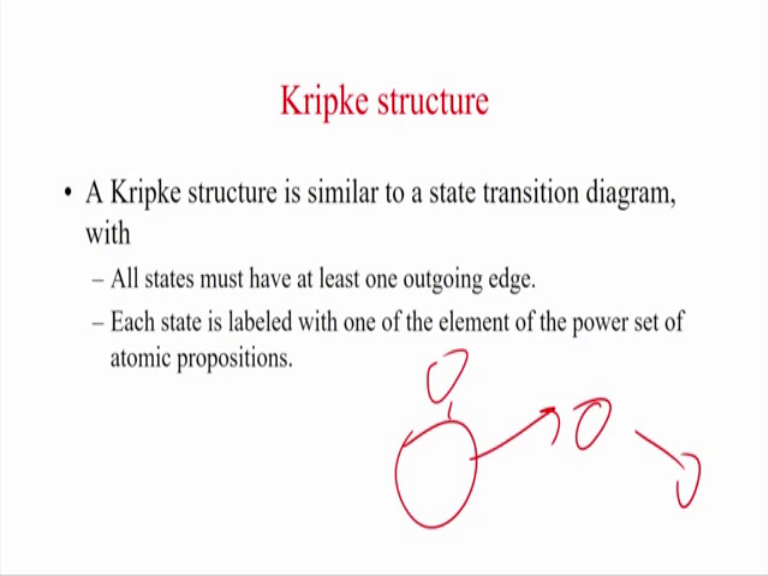
EG means; that means, there is a lot of paths from all in all paths globally f 1 f 1 should be holding EFG 1; that means, there exist at least one path from here from and in that paths specifically everywhere f 1 holds. Af 1, Af f 1 what does it mean? That means, there exists a path sorry Af means for all paths in future f 1 holds; that means, for all any path you take at least one path will be there vary in future sorry all paths these all paths. So, in all paths actually in a future node that is all the path basically f 1 will hold. There exists EFf 1 means among all the paths there is at least one path were in future f 1 is going to hold they are some CTL formulas ok. There are some something called these are called well formed CTL formulas.

(Refer Slide Time: 30:30)



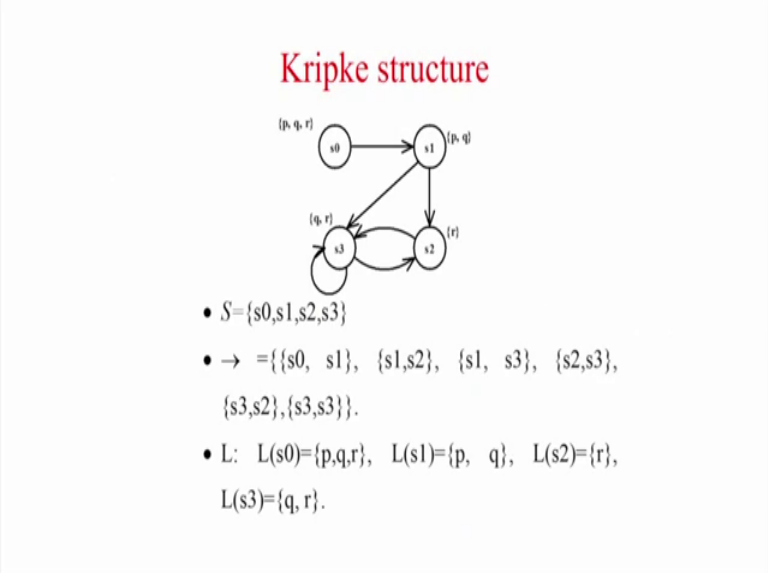
There are something called non well formed CTL formulas because there are lot of I mean you have you have to actually adhere to this paradigm to write this formulas. But what is in this part and what is not available from CTL formula please refer to any standard textbooks or model checking, we all be finding out the idea. Then basically as I told you there is a model on which will be verifying this formula. So, basically there is a Kripke structure we name it like an automata, we called the Kripke structure, which is a temporal structure, where you have the set of states there should be a transition relations, along with that there is something called a labelling function that is very very important. Like just like any finite state machine, you will have state and transitions, but along with that there ios something called a labelling transition. That each of the state some propositional that is your basically your what do I say this basically atomic propositions, like its raining its good eating on such thing heating etcetera has to be labelled; that means, it is true all this atomic formulas are true in that state.

(Refer Slide Time: 31:29)



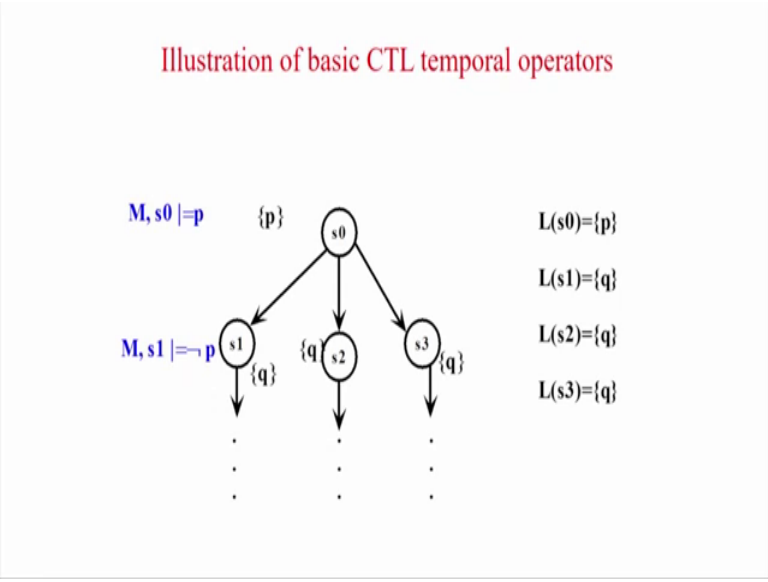
So, basically we always try to write is an automatic atomic form because other after that you can use the logical operations or operators. Like for example, I write its raining and the breeze is there. So, I will write p comma q means p means it is raining and q means weather is good. So, I can write p comma q, because we do not want to write a ampersand etcetera inside the stage. So, labelling over the stage; that means, in a state whatever propositional or atomic formulas are true you can directly label it. So, it is a Kripke structure is similar to a state transition diagram. So, state all state must have an outgoing edge; each edge is labelled with one of the elements of the power set of the atomic propositions. That means, there will be edges, but here what will be power set; that means, any one of any super subset of this atomic propositions, which will be true in this state has to be labelled example.

(Refer Slide Time: 32:16)



So, if you can look at its a Kripke structure and p q and r are some kind of atomic propositions. So, in state s naught these are these two are true in S 1 p and q are true r S 2 r, r is true and so forth. So, that is what is written over here that is the labelling function. So, s is the set of state s 0, s 1, S 2 and s 3, transitions are written over here and L of s 0 p q are; that means, in s 0 these are the three propositions which are true that is p q r are true in this case, S 2 only r is true; that means, in this state p and q are false.

(Refer Slide Time: 32:49)

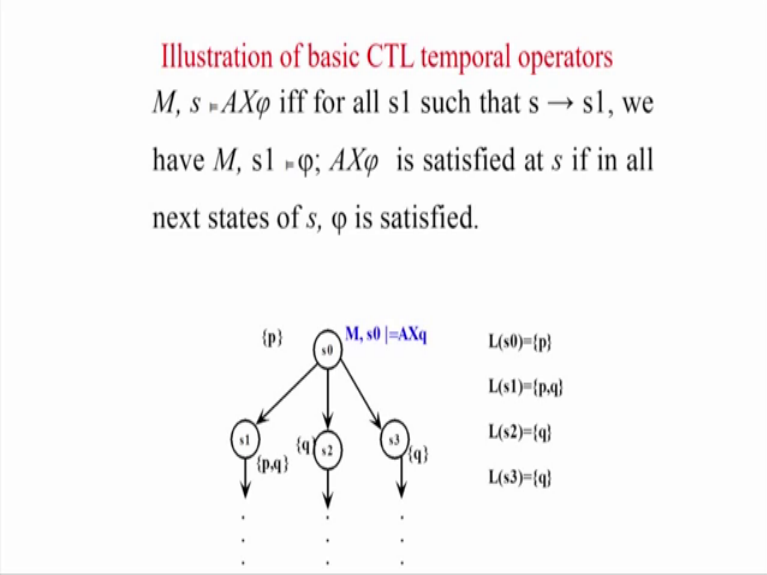


Now, illustration of basic CTL temporal operators, we will start with this. Because and I want to say that in as you can say this is some kind of a model m in I will say what are 2 m.

So, in this case you see this is state s, s naught and whichever atomic property is are true are labelled over here. So, p is true here q, q and q is true over here. So, we can write in m s naught it satisfies p; that means, it s naught p is true that is the way of writing to the properties like m s this is the property and we say that in m s naught is true that is how we write M S 1 naught p is this is one S 1 naught p is true, because p is true over here and here only q is true.

So, we can write at M s 1, p is basically naught p is true. So, this valid in state S 1 because we are not having a p over here; had it had p been there, you would have also written p in this state. So, that is how we do it very simple very simple. If you look at it it is something saying that p p q and they are so, in this s naught S 1 you can say both p and q are true and this property p and q holds state as 1. Similarly S 2 you can write p or q q because q is true over here p is false. So, this an r. So, in S 2 will satisfy this formula CTL formula because as I told you p or q is also one CTL formula.

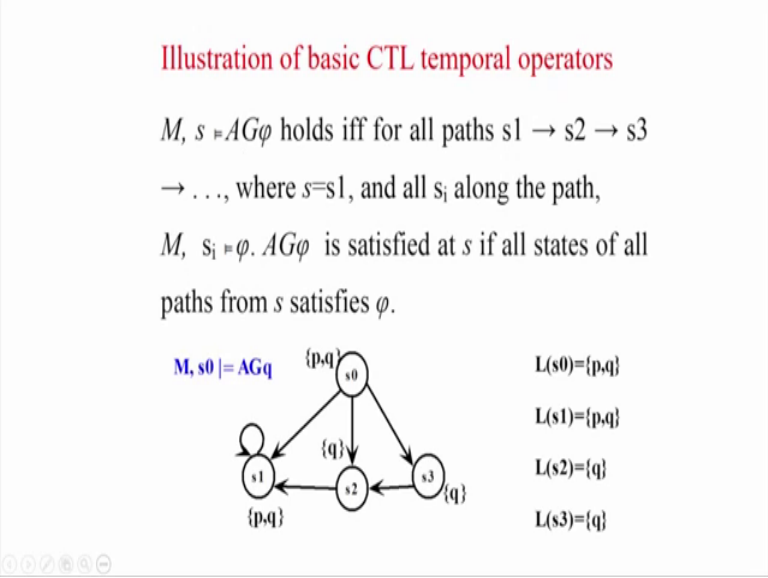
(Refer Slide Time: 34:03)



Now, slightly we doing the temporal business exactly into picture. So, you can see all path next step q. From here you can see that there are three paths 1, 2 and 3 at all this state all this three path from s naught, next step x means next step q is true. So, q is true

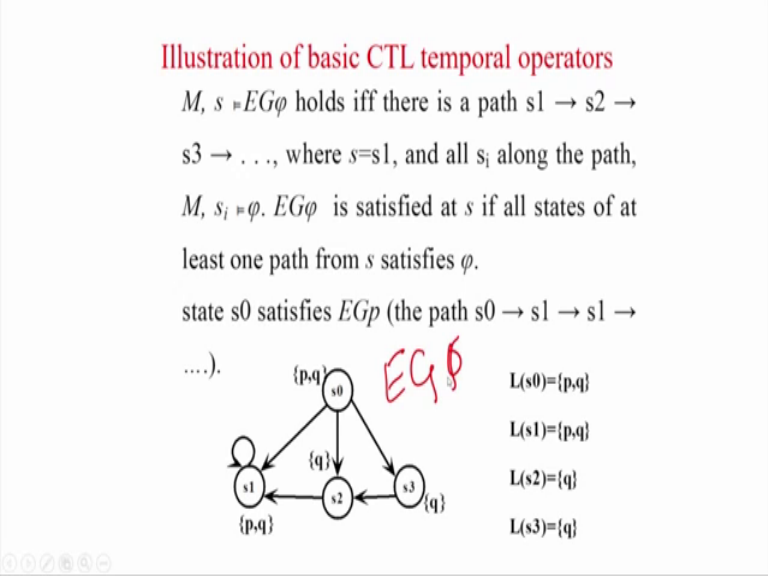
over here, q is true over here q is true over here done. So, s naught will satisfy this CTL property, next one there exist path. So, EX p; that means, there exists at least one path like from here to here, where from here the next state is S 1 when basically p is true, but you can see in this path and this path next state p does not hold, but still the formula is satisfiable because it is not a it is e. In this there exists at least a path here where in the next step p is going to be true. So, this formula is satisfied in state s naught

(Refer Slide Time: 34:50)



Next one what it is saying it is saying that s naught globally, q; that means, from s naught if I take any paths a in all the states p has to be sorry q has to be true. So, you should take a state from here you go over a you keep on looping. So, q is true over here q is true over here its fine. So, I take another path from here I will go to q, q is true and from again q I can loop over here. If I take another path this way, this way, this way everywhere q is true. So, therefore, state from state s naught all paths globally from here whatever paths you take and in all the states in that path q holds, basically this the mathematical definition, you can read this part of the mathematical definition you can read it, but I am just giving you the dreadful meaning ok.

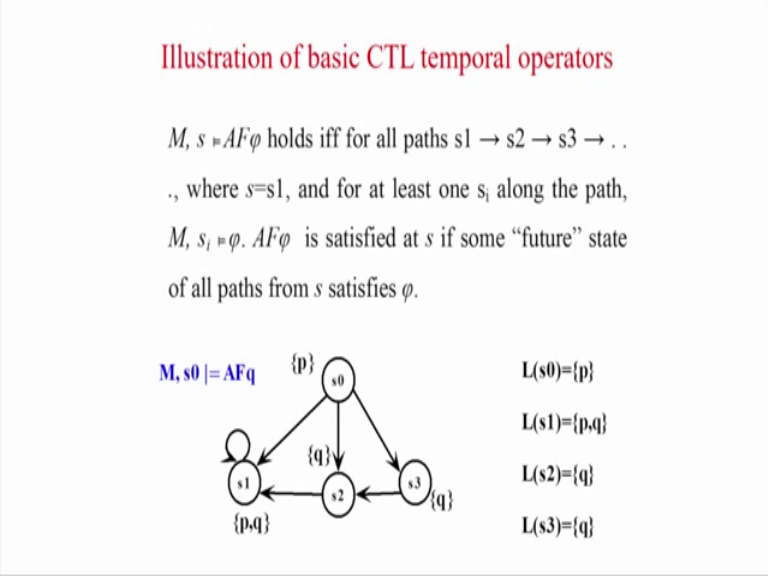
(Refer Slide Time: 35:34)



So, in this case they are having a formula called EG phi. So, what it says? That means, again zoom it. So, EG phi means there exists at least a path, where globally phi globally some phi statement is true. So, if I can say that I am if I make it as a p. So, let me make it instead of phi general p.

So, now, you see. So, it is true over state s naught, because there exists a path from here loop where globally p is true. So, p is true over here p, but I cannot write there exist for all path a g, I cannot dissatisfy, but I cannot write a g. So, I cannot write AG if I write AG it will not be true. So, if I write AG what is going to happen. So, AG means for all paths globally p is true, but there is one path over here or there is at least one path like one path from here to here, p is not true there is a p not true. That is how AG phi p will not be satisfy that s naught, but EG p will be satisfied.

(Refer Slide Time: 36:22)

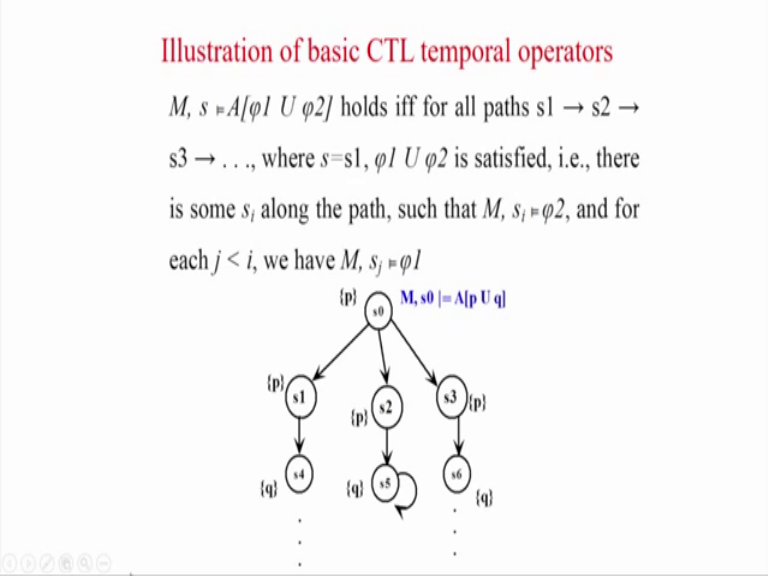


But I can write something like for all paths in future q is true, that is holding in state s naught why because see in this case in future. So, in future q is there right. So, here state will be there in future q is going to be true. There is a path over here in future q is going to be true. There is a path here where in future q is going to be true.

So, future means what it may be for all path basically, at present basically I can also slightly modify this then also it will be holding. So, if I make it as sorry right. So, you can see here. So, it says that in all path in future, q is true. So, here p q is false, but if you take a path here in future p is q is going to be true. In this path again q is going to be true in future; again in this path another path you take q is going to be true in future. Even if nothing is true over here, that also will satisfy this because in this case p not true, because I need q to be true in this path not true, but here actually q will become true.

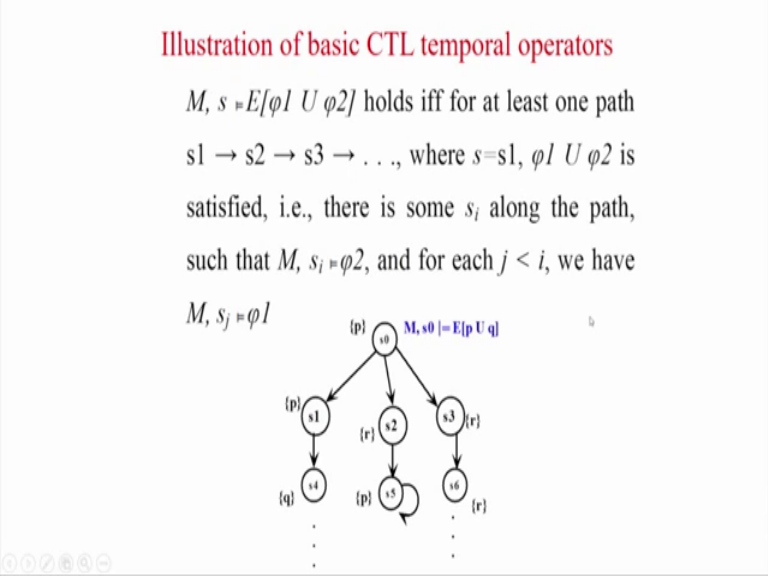
Even I think it will stop even hold, if I even destroy this because there is the path here where both p and q are true that is in future not globally is. In that from all paths here in future q is going to hold and in fact, that is holding over state S 1 ok. So, this as very simple way of actually representing some of the formulas and I am giving some diagrams which are telling you where it holds and where it does not this is again representation of p until q.

(Refer Slide Time: 37:58)



So, in all paths from state s naught p until q. So, p hold p stops holding q starts, here p holds here p holds, p stops q start taking over. So, in that path also so, here all the three paths you take, p is holding, but whenever p stops holding u takes over. So, it is true over here.

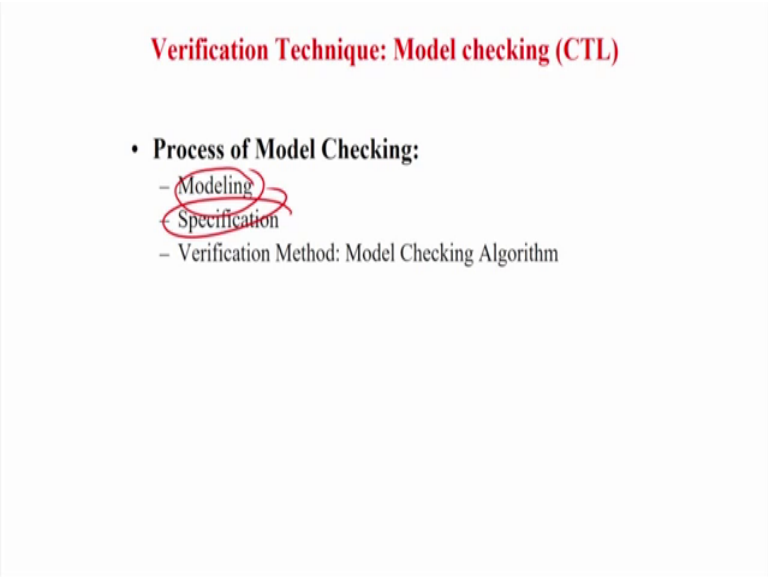
(Refer Slide Time: 38:21)



In this case if you slightly modify diagram, so, in this case if you see p stops holding, but q does not hold over here. So, in this path p until q is not satisfied; similarly p here p does not hold here q does not starts it job. So, in this two paths basically this path and

this path p until q does not hold, but this formula if you see this thing e, there exist at least one path where p until q should hold and actually its happening in this case p, p whenever p stops q takes over. So, in this case it is an existential it is actually satisfied over here.

(Refer Slide Time: 38:50)



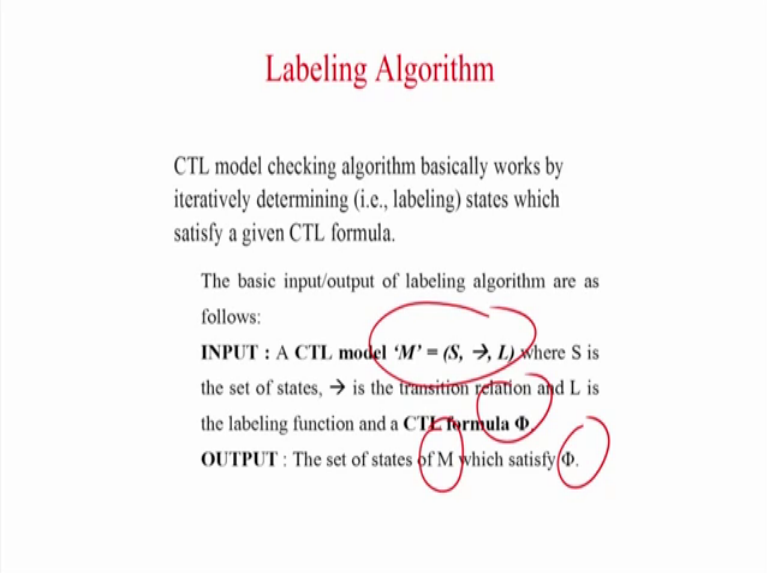
Now, what is a how a model checking is basically done? So, first is modelling. So, like this things basically as I told you this structure you are looking at it the Kripke structure this is the module. So, by understanding the system you have to make a model. So, as I told you it is mainly a two way approach, sometimes what happens sometimes the officially speaking or formally speaking every designer should first have a model and then they can go for implementation, but in industry that does not happen always that way.

So, this a hybrid kind of approach you implement something you model something because finally, implementation is already deadlines are there. So, just as we are doing in our btech life, same thing peoples also are doing in a industry in a slightly coat uncoat similar level they also have deadlines to meet. So, when deadlines come they do not know exactly go for real kind of modelling they exactly try to fix up something and implement and bring it to the market. These are real fact, but I mean this also should be taken with the coat uncoat pitch of pinch of salt, because this is you cannot say that it is universally true, but for many many cases it actually happens that way. So, basically that

is what happens after the product has been made people or or in the final state of design verification team, basically understands a design and they make a formal modelling out of it that is the Kripke structure, then there is something was specification that is the formula in all paths globally this should happen.

So, what should happen basically you have to write in terms of formula like for example, if you say that you have designed a printer then you have to write for all paths globally, whenever I request those in future grant should be accepted. So, that way you write your design intent in types of specification and then there is something called a model checking algorithm, which will try to find out in which states the formula is valid and which state is not valid. So in fact, no fi you are giving some kind of specification to be checked, there should not be any states in the system where the fast specification is invalid if this invalid that is actually a counter example and you have to verify what has gone wrong. In this case what happened we were doing it manually ok. So, in this path it is not satisfied in this path is not satisfied in this path is satisfied. So, we are saying that E p is existential. So, it is true, but if it is a large system of course, you can never do it manually. So, that is actually the idea of model checking which is automatically do this labelling for you.

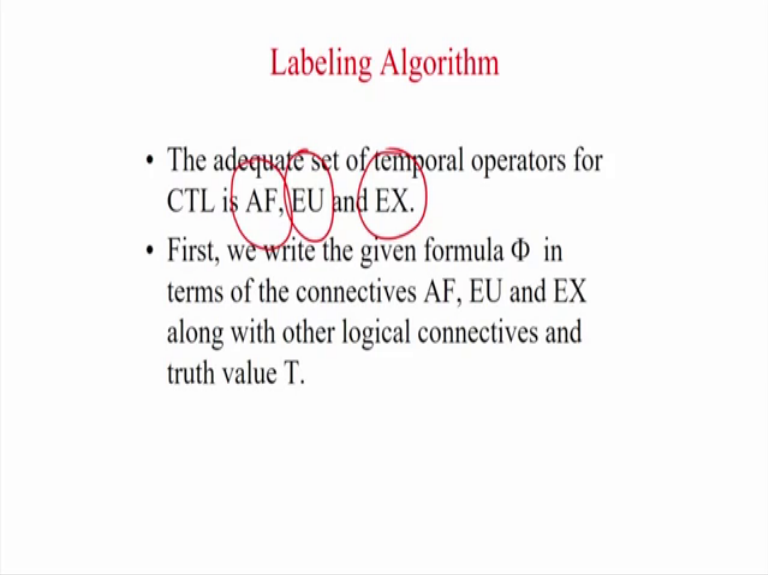
(Refer Slide Time: 40:56)



So, now you are going to see model checking is basically something called CTL model checking something called labelling algorithm, which we will now look at. So, basically

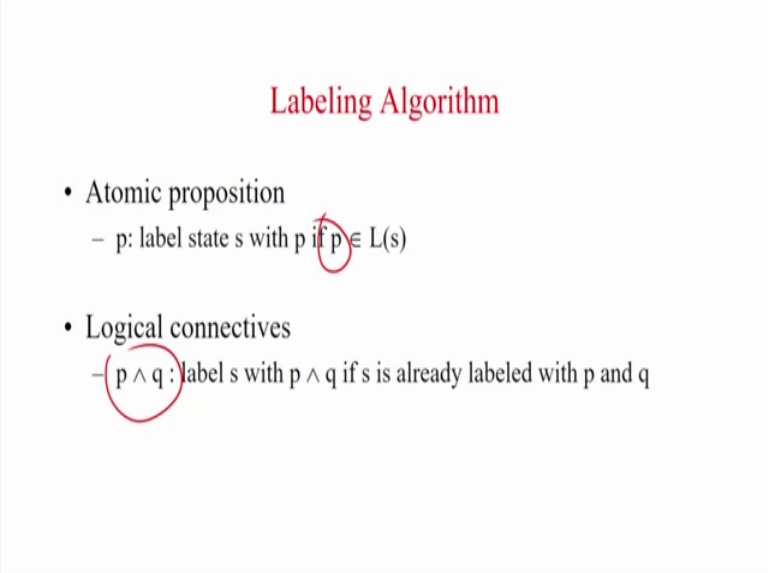
what it does it takes a CTL model M, and it takes a formula phi and output is the set of states which satisfy phi. So, generally your formula to be satisfied in all the states and whichever state is not satisfying, it will give a counter example that you can go for and debug it. So, basically there are lot of CTL formulas can be there, but generally all paths in future there exist a path until and there exists a path X.

(Refer Slide Time: 41:21)



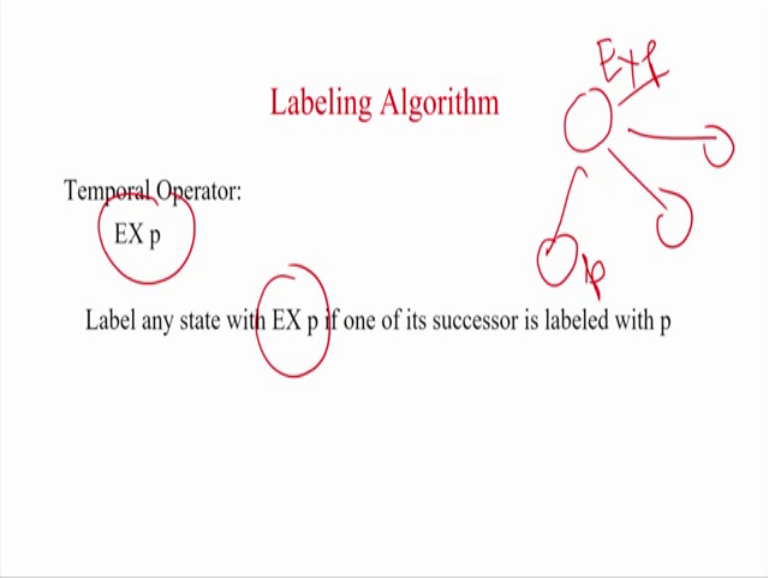
So, basically this is a adequate set of operators for CTL. Any other CTL formula can be represented in terms of this. So, as I told you in the in this lecture we are trying to put everything together, in (Refer Time: 41:40) you are not going to show the proves etcetera, you can look at any standard textbooks or model checking which will tell. But these are the three most fundamental property fundamental temporal operators for CTL, you can write any CTL formula using this. So, in this lecture we will see how the modelling or model checking is done for on this basic adequate set of temporal operators.

(Refer Slide Time: 42:00)



So, atomic propositions already told you p. So, if some states you have to find out whether the atomic propositions is valid or not it is just simply you check, whether on this state whether the labelling is with p or not. Logical connectors are very easy to check that is if you say when all the states p and q should be valid. So, you have to find out in both the states the labelling of p and q should be there.

(Refer Slide Time: 42:20)



Now, we are going to see when are going to real temporal business. That is there exists a path where in next state p as true, how do you actually do it? Label any state X with p x

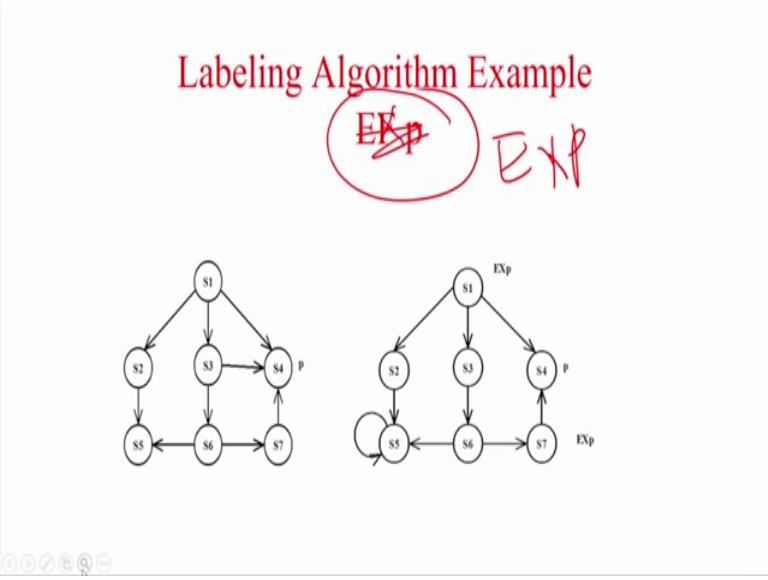
the algorithm is label any state. With EX p if at least one of its successors labelled with p; that means, if there is a state x and may be assume that there is another state here may many many paths may be there here at least, it is marked with p. Then you can easily mark this as EX p and you have to keep on doing it there is until there is no other change. That means, the model is there, you have to drowse through or you have to traverse the entire graph and basically if you find out that any state where p is true with that state you have to mark it as EX p. And you have to keep on doing until there is no more change or in fact, you need to traverse the entire the graph then you have to do it.

Now, complexity comes in. So, if you can find out you already know automata theory, that if the number of state variables is n the number of state states in the model can be 2 to the power n. So, if your system a 100 variables, these state model or the formal model on which you will do this label or model checking will be 2 to the power 100 no way you can achieve it.

So, what you are going to see today basically is that that modelling change modelling change. Generally you are going to see, but along with it also we have to appreciate the you will appreciate the fact that is a very large system you cannot have an explicit state warning on which we have to do the verification. So, in the model itself is not there, I do not know where you will do the verification.

So, all the future lectures will try to see how we can reduce the modelling or the model itself so, that you can do verification on the reduced model. So, we will see binary decision diagrams, we will see arithmetic decision diagrams and so, forth right. So, that that modelling is very simple so, I mean. So, if I mean if there is some state EX p is done in this manner. So, graph is there some state p is there the beforehand state or the root state, I mean the predecessors state you can mark it EX p.

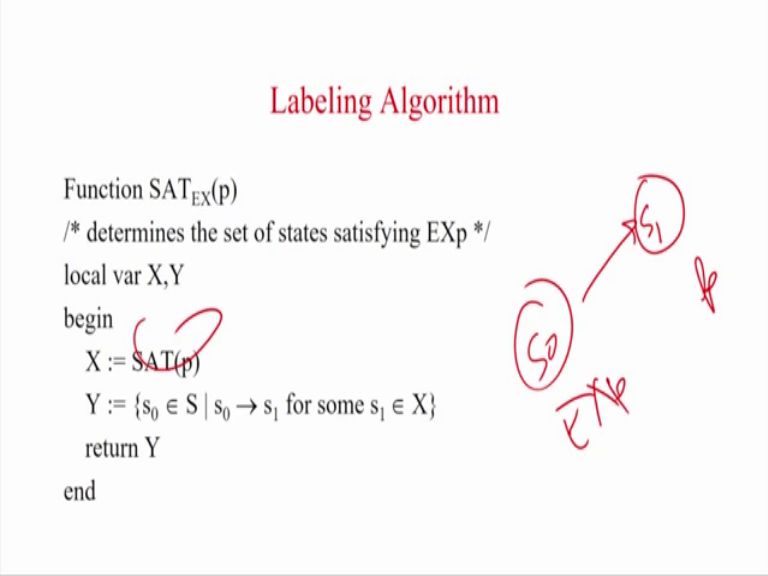
(Refer Slide Time: 44:11)



Now, so, this is an example. So, that is next state now another example we are looking at EF p. So, the three ones we are taking. So, EX and then EF. So, although EF is not a part of this, but still we are just giving a the example. So, sorry sorry basically it is e x I should not have written it as this one I should have written it as EX p sorry.

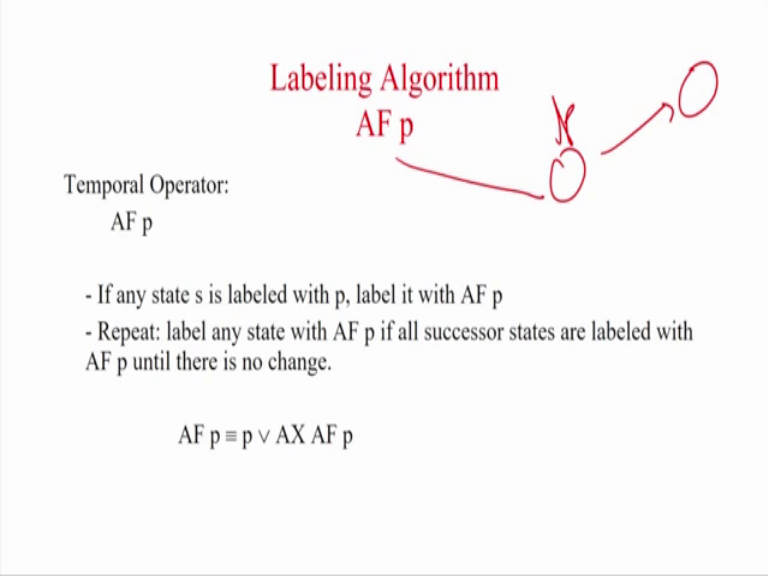
We need not talk at all about EF p because it can be represented in using other properties. So, how do you do it ah. So, in this case p is there. So, you browse this whole graph and you can find out that the mothers I mean the predecessors state is s 7and s 1. So, very easily you can write EX p here EX p here and your basically job is done. So, only these two states basically you can see the next state the property p is true. So, that is actually a model checking. So, if somebody says that EX p has to be the very very important property and it should be done in all this change, then you will say that the only in S 1 and s 7 is valid, and in no other state is valid basically that is your counter example see what has to be changed to make the design proper ok.

(Refer Slide Time: 45:14)



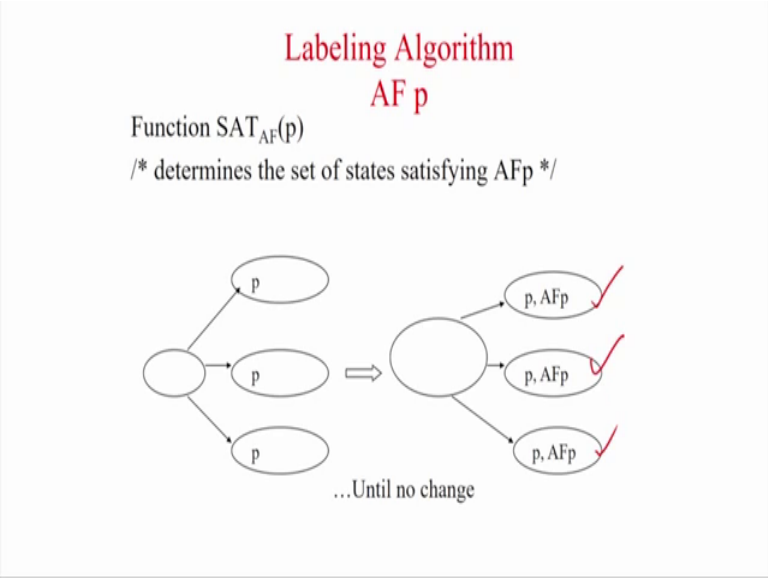
So, we write it this is the formal way of writing the algorithm. So, SAT p that is satisfiability of p ah; that means, in whichever state p is true that you have to mark. So, sat of p means only in S 4 p is true that is equal to state set x, and what state has to be labelled some transition s naught to S 1 and in S 1 basically x is true that is nothing, but here this property is true. So, this one has to be labelled. So, basically if you look at it. So, x is somewhere where p is true, and you have to make set y, which is which is nothing, but all states s naught such that there is a transition from s 0 to S 1 any S 1 basically p is true that is saying something like s naught to S 1 in S 1 we already know p is true and by this sat formula, then this one you have to labelled it with that e x simple way of doing this a algorithm way of writing it.

(Refer Slide Time: 46:08)



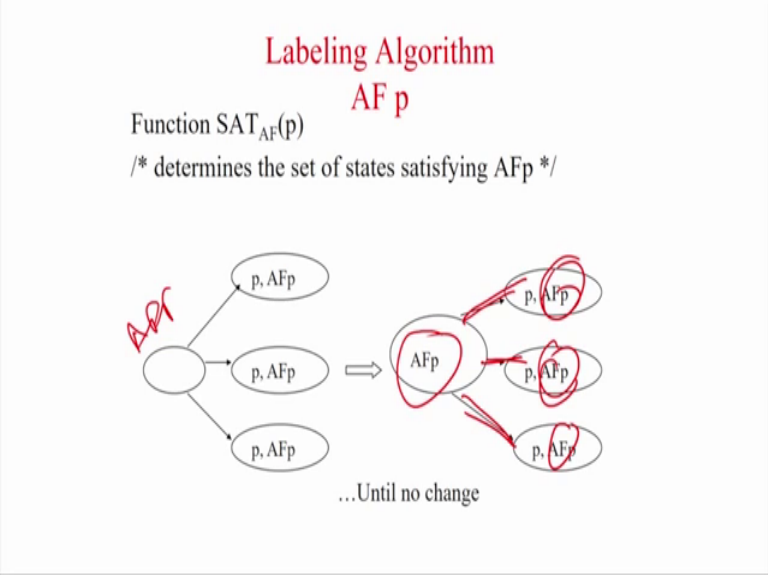
Now, in all paths in future. So, we are going to take AF now. So, how to do it? In this case it say in all paths in future p has to be true. So, and we always again I want to highlight this fact that present is future. So, if I say that here p is true, you can write that this state satisfies AF p; that means, present is future. So, in future it may be true, but if in the present state itself p is true; that means, it is valid. So, basically what do we write the algorithm is, if any state s is labelled with p labelled it with AF p as I told you present is future. And repeat label any state AF with p if all it successors states are labelled with AF p until no change, I will show with an example. The formula is p if p holds in the state itself is true you just label it or A X AF p that is all states next all paths in future p. So, that is the way of labelling let us not go too much in the formalism you can easily understand after I show you the example.

(Refer Slide Time: 47:04)



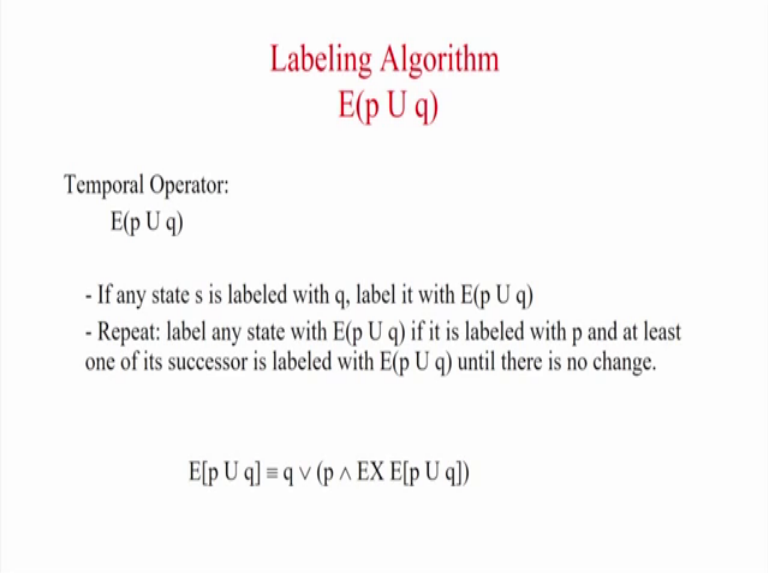
So, wherever p is true you make AF p true AF p true AF p true.

(Refer Slide Time: 47:11)



Then then basically you put it this way, that is now you see here all the states from here it is labelled with AF p, AF p, AF p assuming this an assumed to be a situation with a leaf nodes. So, only p is there. So, I am making with p AF p. Now you can see I can also mark it AF p because all all the paths here AF p is true. So, in this case you will find out that this one AF p is over hand and similar on this case, until there is no change or we need a fixed point.

(Refer Slide Time: 47:38)

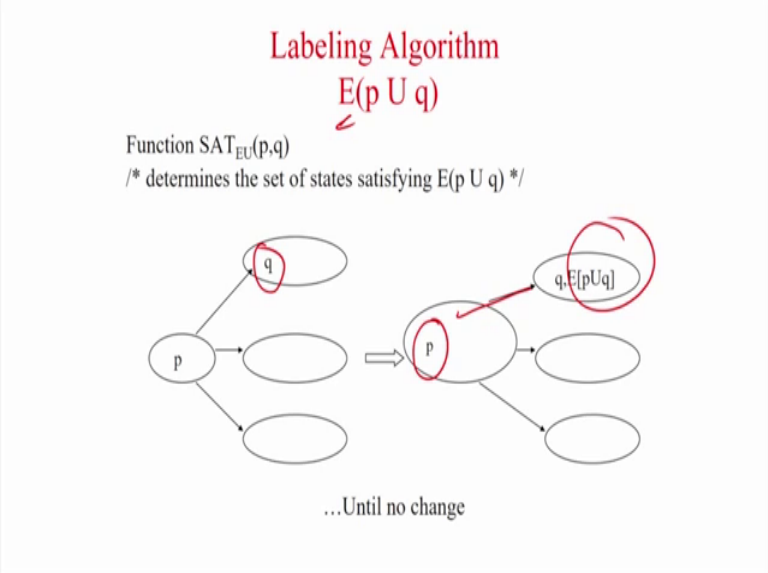


Now, we will go for the temporal operator. So, by doing it actually tell the same thing. If some state is labelled with this mark with AF p, label any state with AF p if all successors states are labelled with a p until there is no change. So, if there is p just do it because present is future. So, even if there all paths any paths are there it can happen also, but I can still mark it in all paths in future true, because present it has been true over here. So, even if know where p is true in sequence I do not bother because present it has been true. So, I will mark it as AF p and then finally, we can also make it because all the senior I mean predecessor nodes are marked with AF p done. So, that way you can do the modelling.

Secondly, this is e there exist p until q. So, how will you do it? If any state s is labelled with q labelled it with, there exist p until q because here q is the safety it says that p should hold until q starts holding.

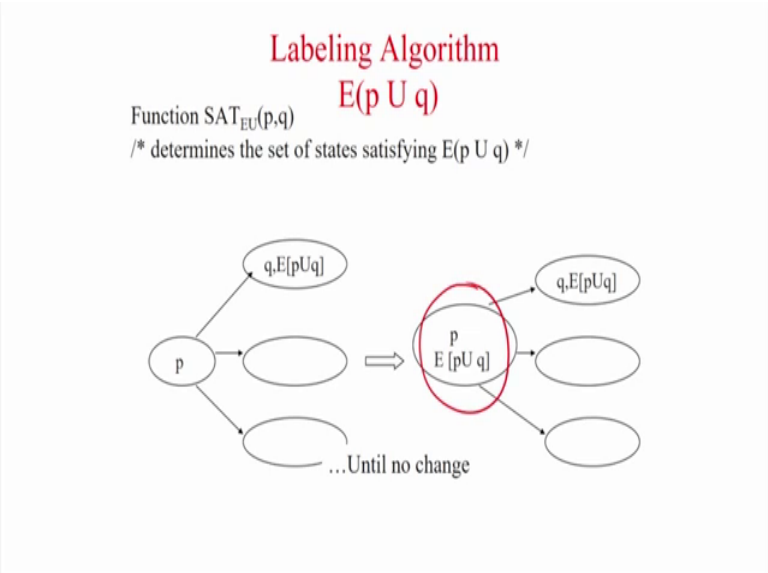
So, if am I q starts holding, you definitely put it. Then similarly label any states with this if it is equal to p and at least one of its successor is labelled with this until no change ok.

(Refer Slide Time: 48:45)



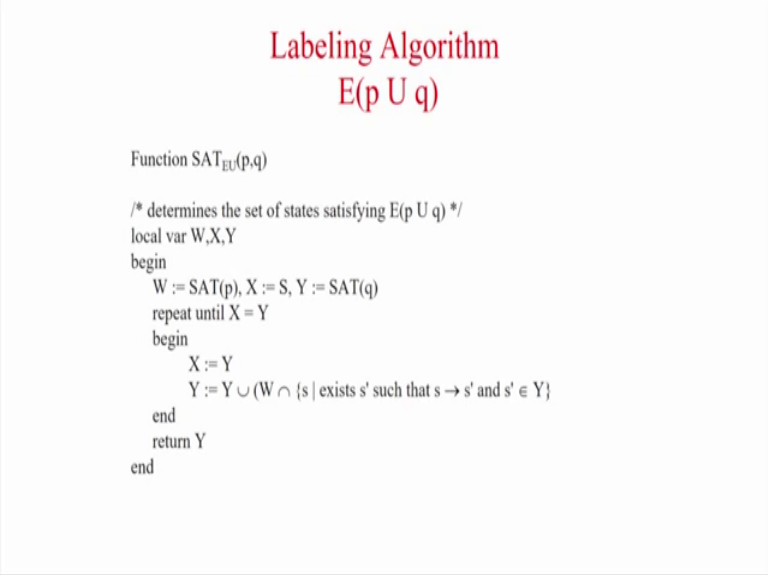
Always better to show the figure. So, here q is equal to true. So, if q is equal to true; that means, basically you can directly write E equal to there exists a path where p until q, because here q has become true. So, you need not think about at least even p p over it, because q is starting to be true in this case. Then what it says? There at least from this place p is true first of all, and there exist at least one path because we are looking at the existential quantification.

(Refer Slide Time: 49:15)



So, at least one path where p until q is true. So, there also you can put it p until q and you are labelling. You have to keep on doing it, I will have fixed point is this or more change fixed point there is no change. So, basically that is nothing, but if you read it, it will be this label. Any state with q, label it p until q that already you have done with first case, label any state with p until q if it is labelled with p and at least one of its successors is labelled with this. So, that is p is there and this stage is already labelled with e p until q. So, we have done this right.

(Refer Slide Time: 49:43)



This is the algorithm, I am not going to go for that, what I have told you is return in this pseudo.

Now, one more we are going to see. So, we have seen how many so, we have seen basically AF e u and EX ok.

(Refer Slide Time: 50:01)

One more you are just going to just see this one, this may not be AG, AG p is not actually a part of this basic set, but still just we are going to give you an example; that means, the idea is that for any operator you take like e e e e p until q or a for every such formula will way of labelling algorithm will be different.

But actually the why I discuss mainly for this three, because any property you write can be represented in this terms you can you have to modify then in this terms then you can write. But it does not mean that there is if there is some temporal operator like this one AG p, which is other follow basic set cannot be model checked they can also be model checked and there will be a different model labelling algorithm.

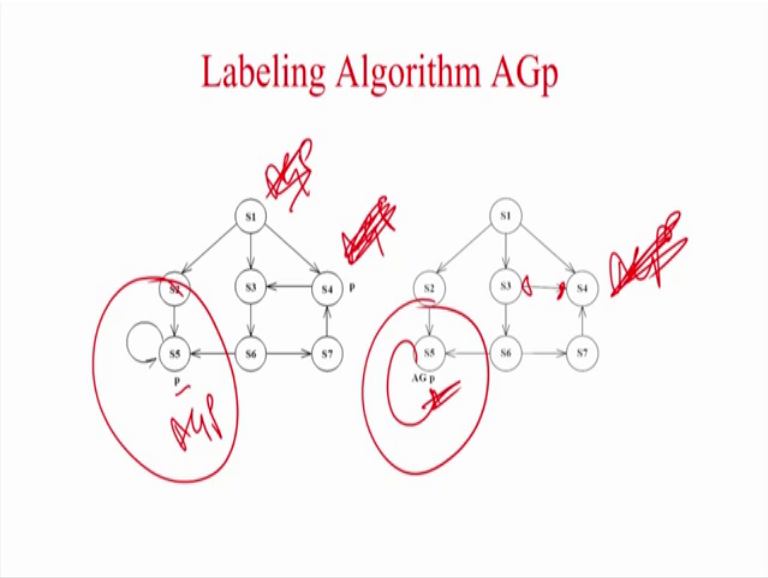
So, any temporal operator you take in CTL, there will be different model checking or labelling algorithm, you can use them to do it. But in this case we have first limited to the preliminary three basic three ones, because like AG p can also be return in terms of this three basic operators. But still just to for the sake of completeness basically I am also going to show you that even though this is not a part of the basic set still how to go for a model checking for this.

In fact, we always write different types of CTL properties and it driven always try to model there in terms of basic ones. No we do not do that we try to keep it in a natural form like also we can use something which is non basic and we can have and we have different model checking rules for them and we can do it, but as I tell told that this as

compress lecture. So, in this case we have tried to slightly deviate a bit and we have mainly shown for the three basic operators, and as a slight I am also going to show that even though is not a basic operator, but still how to go for a model checking for this and the algorithm will be slightly different.

Design an AG p that is all path globally p; that means, everywhere p should be true how to do that? So, first you label all states with AG p, you just forget with the p is there or not you remove it. If any state is not labelled with p delete AG p, and keep on doing it delete the label AG p from any state if all its successor are not labelled with AG p a and there is no change. It is very simple like first you label all this steps with AG p and then start removing where we have made a mistake. This slightly different way of model checking the algorithm is slightly different compared to the other three, other three basically you were adding up the labels here we are deleting the label.

(Refer Slide Time: 52:16)

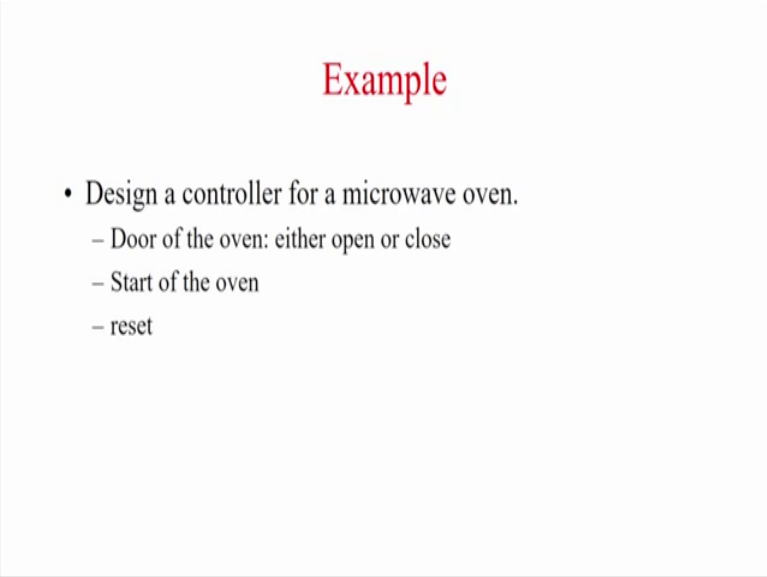


So, for example, if you look at it. So, in this case here p and p will be true. So, directly we will write here AG p and then you have to keep on deleting it. So, you will find out that in this case if you look at it, I will do it, I will do it in all this state's basically just I am just giving the example, I will do it all the states AG p, but in this case you see in this AG p. So, there from here all paths globally p has to be true now if you see look at it here p is not true.

So obviously, I have to delete this. Similarly all paths will have AG p and everywhere it will be deleted because there will be at least some states in future, where AG p is not these not there or AG p is not true in that case. But if you look at this state specially sorry there should be a loop over here therefore, got to draw. So, in this case if you look at it. So, there is a self loop over here. So, in this case if p is to the successor is only p S 5 is the successor of itself only state successor which is AG p is true. So, these state AG p will be true over here; that means, in this state all paths globally p is true, because this is a self loop over here. So in fact, what we have done all this states will first label with AG p including this, and then we will find out that there is a path like for example. So, in this case you will find out that this is a state successor with where AG p is true is not labelled because p is not true over here I will delete it.

Similarly, all this state's label will go, all is the remaining here will be there because its a self-loop and AG p true over here done.

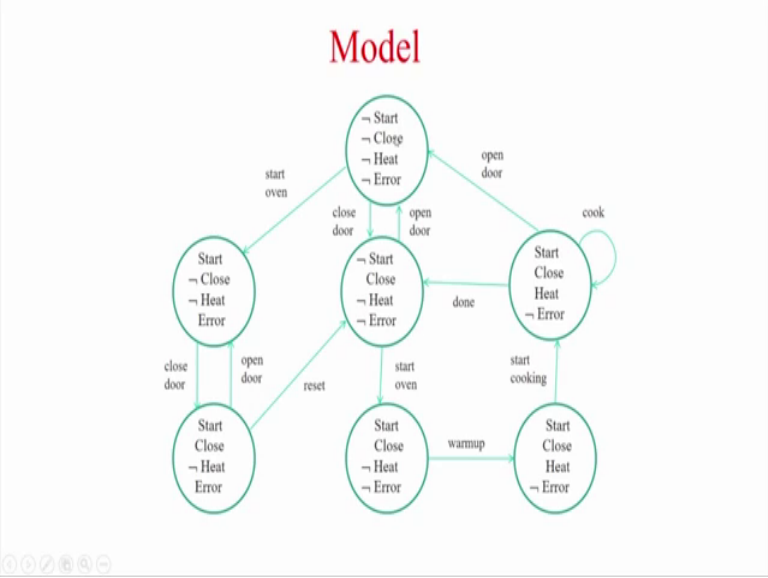
(Refer Slide Time: 53:40)



So, now we are to coming to practical examples. So, lot of stray examples abstract examples p q r state machines we have given, now basically we are coming to a practical example about microwave oven control and see how basically CTL model checking is applicable over here. So, it says specification design of a controller door of the microwave either open or close start the oven reset; that means, the door can be open and closed, there will be heating and non heating and some reset and some designs will be

there. Then (Refer Time: 54:08) microwave what we do we have to place something close the door, put it will be heating up and then actually we will stop it open the door and, but one thing has to be very much important we have to remember that. One very important thing that when the door is open, you cannot have the heating on that is a critical catastrophe will happen that micro radiations will come out, if the door is open and your heating is on. So, that has to be there. So, this is basically your model.

(Refer Slide Time: 54:35)



So, again if you can see there are some variables and like start close heat and error. Start means on close means door close, heat means microwave is on error is some variable. So, 4 variable is 2 to the power 416 possible combinations are there. And if I make an explicit enumeration, you can find out this will be a huge state space that is where actually the complexity is going to kill you. So, in all the optimisation lectures basically we will see that how to handle these complexities by not explicitly representing as a explicit state model state space model.

But anyway in this lecture we are not going for optimisation, we are just trying the basics. So, we are so, into see how model checking is basically done on this, microwave control. So, it is saying first it is not start, not close which door is open not heat oven is off not error there is no error is off and the door is closed not closed means door is open door is basically open then if you can look at it here you can find out something

interesting that it says door close means the close will be true door open means, the it will be true now if you see what is the flow.

So, first you will close the door then you will start the oven. So, basically start was not start it just becomes start right then it will be warming up; that means, heating is on then basically cooking means when heating is on it will go over here same state and it will it will be a looping because cooking is going on once it is done.

You can find out you can go over here heating will be off and then again you can open the door or basically you can once the cooking is also going on, you can directly open the door, which you always do in microwave oven sometimes we do , but immediately at that point if you can see that it will become off door sorry if you open the door. It will be off heating will be off and the door is going to be open; that means, this path basically if you look at it cooking is going on if, I open the door it is going to be a automatic off that is start no start open no heat.

So, that is a safe condition so, this part actually if you models the normal operation of a heat your microwave. So, this part also you can see sometimes you can take this path or sometimes I can take sometimes you always do it heating microwave is going on, quickly want to have something open the door bring it out. That means, then you are going from here or you can take the normal procedure, you say stop heat will stop and then you can open the door, that is switch off and then open sometimes you can directly open. So, anyway, but the safety feature is already given.

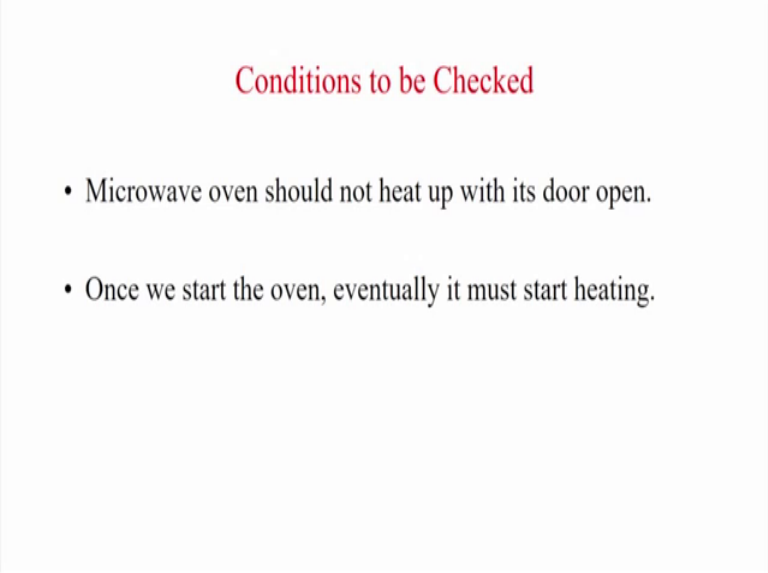
Now, the other part is basically an error part. So, some error are bit will be given. So, your door is not closed that is open. So, your door is open and you start the oven. So, basically heating will not be there and it will that is your door is open you start switch on the heater. So, it will say error, that be some beep will be given and then what you can do is that, basically in this case you can close the door and then you can again and reset it.

So, you close the door then automatically will be reset and your and basically you can start doing the normal heating point over here, there is one way of doing it. So, basically start the oven, the door was open then basically you close the door then automatic reset and your come to the normal track or again you can open and close the door nothing is going to happen. So, this part actually this two part basically I am look at it here, open

this is something like its a wrong way of operating the oven, but still it works with a beep.

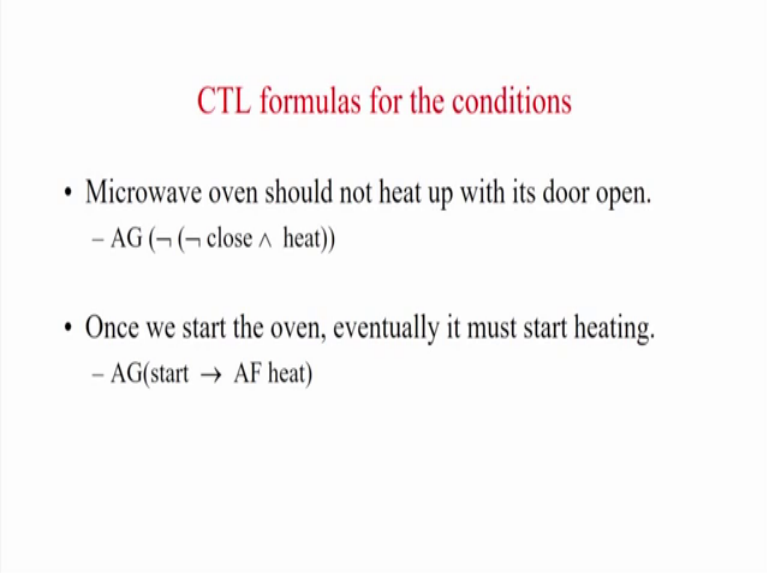
So, for example, you might have kept the door open or slightly open your door node starts switching on for heating. So, it will be a beep, then basically you close the door and then basically automatic will reset and start your basically oven will start. So, you can easily look at it and you can easily interpret what I mean to say right.

(Refer Slide Time: 58:05)



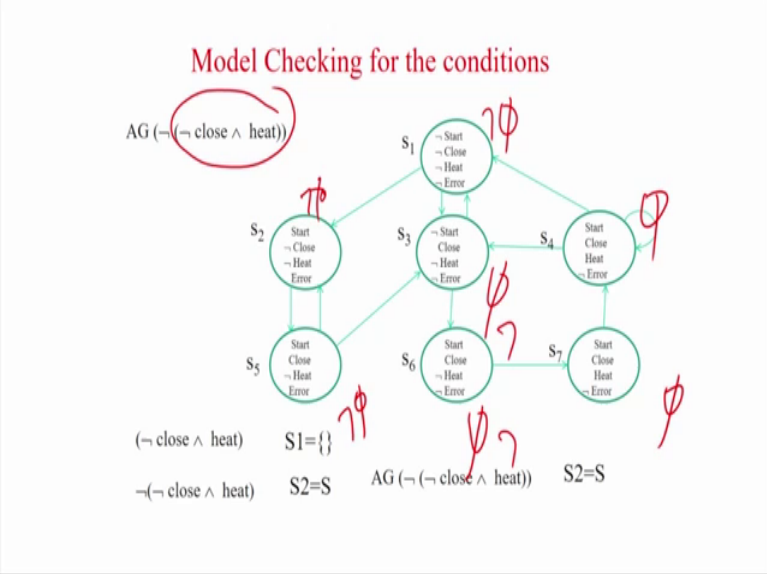
Now, two conditions to be checked. One is microwave should not heat up with its door close very important. Secondly, once you start the oven eventually it should start heating I will show both this specification, how they are to be model checked. So, what it is saying?

(Refer Slide Time: 58:19)



First one is saying your microwaves should not heat up when is door is open. So, open means that is not close and heat this should not happen so; that means what? It is basically something like not of this condition is bad door is open heat is done. So, that is bad. So, you should have a not of this and you should hold globally in all this state AG. And the second property is saying that once you start the oven eventually it should start heating; that means, if you start it in all paths in future it should start heating that is something we all request for all machines. We switch it on always in future all the paths heat should start or the machines, would given and it should happen in all paths globally.

(Refer Slide Time: 58:58)

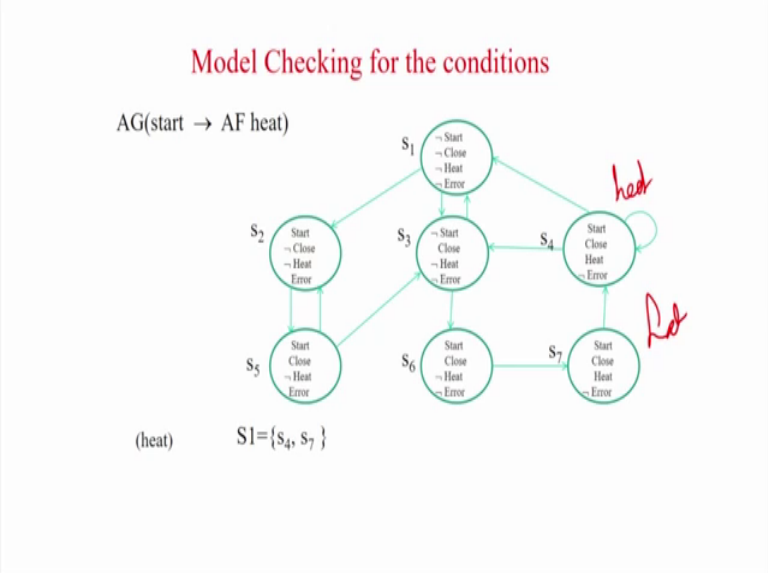


Now, I will start doing the model checking. Again first one is very simple first one we are saying that not close and heat that should not be true in a state. So, if you can look at all the state in no state it will be true that is not close and heat this also not true over here because not close is true, but not heat. So, this combination like not close and heat you will find out that, it is not true in any of this states; that means, first we will model check this is very simple you traverse all this states and just check if any combination is there or not.

So, this combination is not there. So, this combination basically let me call it phi. So, it will be phi phi label it phi phi. So in fact, not phi will be true in all the cases, so in fact, I did not label it. So, in none of the states this phi is going to be represented it; that means, in everywhere not phi is going to be true. So, let me call this whole thing not phi as some phi not of not this one that is the full formula inside I am calling it as zhi this whole thing I am calling as zhi.

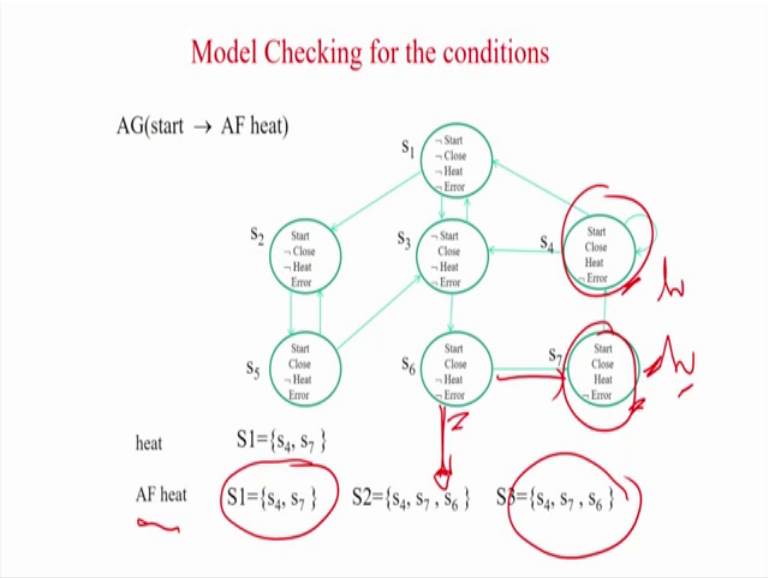
So, this one will be true over sorry. So, it one will be true over all the states all the states, because no where this combination is true this combination is true nowhere. Now all paths globally, we have already seen the labelling so. In fact, as all the states basically has the zhi true. So, AG zhi will be labelled in all the states and basically AG of zhi is going to hold in all the states. So, you will say the states basically satisfies this formula and so, this model is verified to be true for this. That means, there is no state that is that is in this case if you if if the door is open, the micro wave is not going to be heated. So, very good safety is done.

(Refer Slide Time: 60:40)



Now, we are going to check for the more complex this thing, in which case we say that I put a start it is very critical put a start and in future heat should be there. So, what I have to do it in steps. So, what I have to start thing about in which states heat is true that is heat I have to first check. So, if you look at it you can find that heat is true in state is 7 and S 4 only. So, S 1 means heat s S 1 and S 4 it is true. So, heat is true in this case. So, I am writing it as heat. So, some heat is true over here that is S 1 right.

(Refer Slide Time: 61:13)

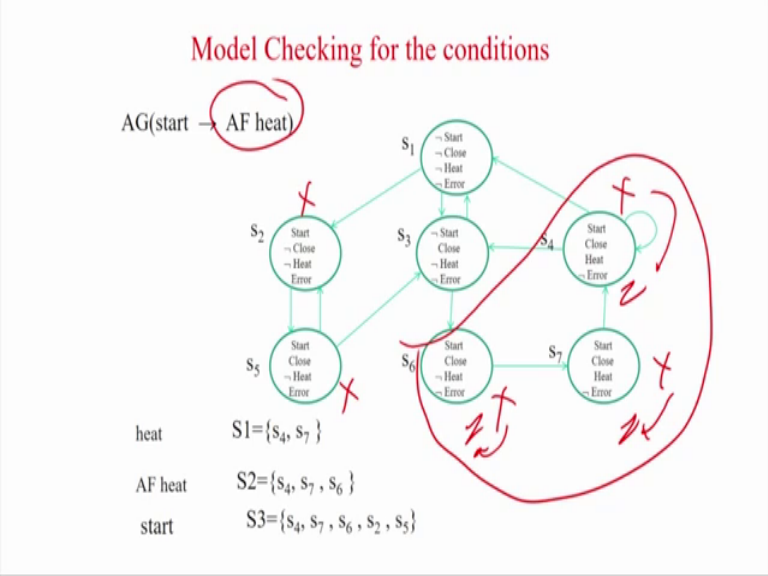


Now, all paths in future heat. So, basically so, these are these are the two states I am marking S 1 sorry S 4 and S 6 which are true with heat. Then now in all paths in future in which most state is there. If you look at it I will doing the same model checking algorithm for all paths in future, that labelling algorithm you have to do if you do it you will find out that S 6 will be coming all paths in future heat. So, this one heat so, in all path. So, these are the two states were heat is there.

So, now I am saying all paths in future heat. So, if you look at basically S 4. So, S 4 right just a minute ah. So, S 4 and S 6, S 1 is the initial conditions. So, these are two states where heat heat is true now basically you see in which path all path in future right if you look at it if you look at your state S 6. So, S 6 there is only one path from here in which in this is heat if it is true if it is true over here. So, from S 6 there is all path is only one path here where in future it is going to be true.

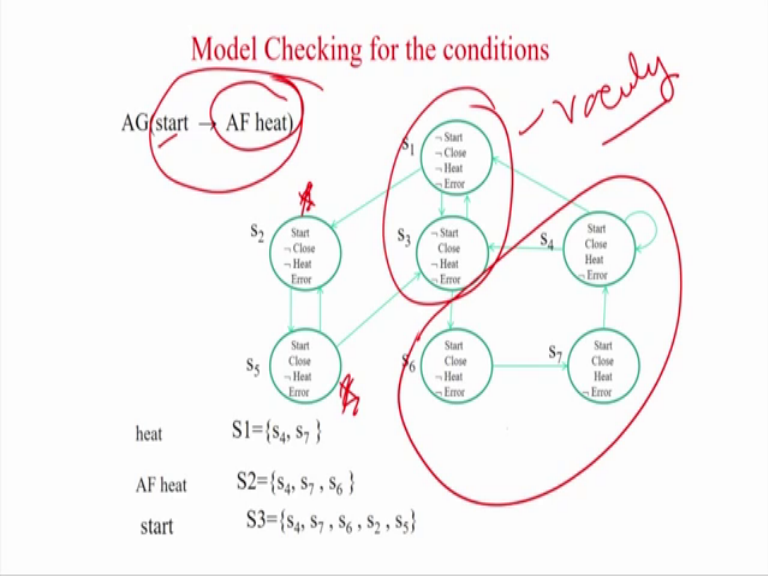
So, straightway S 6 will enter this group now, but you can easily appreciate that S 3 will not go into the loop because S 3 has one more path which is looping over here where heat is not going to be true. So, S 3 S 1 will not come into this group now so, S 4 S 6 and S 6 now if you look at S 4. So, this S 6 has come over here S 7 also will satisfy because as I told you all paths in future is present also. So, here automatically heat is true. So, automatically S 7 will be there and here also it is true. So, by default as heating is true in my parent state automatically this will be there. So, this is the only new state which is coming in. So, all paths in future heat will be S 4, S 7 and S 6 these are the two states by default because they are home state for heat and this is one more state is coming in because this is only one path from here where in future heat is going to be true. So, finally, all paths heat means these are the three states you should mark this one, this one and this one where this is going to be a true right. So, now, in this case. So, this one, this one and this one are the three states where this going to be true.

(Refer Slide Time: 63:23)



Now, I have to now check for start where start is true. So, start is true basically if you look at. So, I will make a cross. So, start is true over S 4, S 7, S 6, S 2 and S 5 these are the states where basically start is going to be true now x imply b. So, now, the states where is very important, but in this S 2 and S 4 this AF heat is going to be false. So, in these are the two states where AF heat is false, S 2 and S 5 AF heat is false now as I told you there is something called vacuous business. So, this three states; obviously, this going to be true, because start that is start implies AF heat so, this one is true implies this one is true implies this, this one is true implies this. So, of course, in this three states S 4, S 6 and S 6 you will find that start implies AF true is holding. So, I making a chart now in this S 2 and S 5 if you loop start is true, but AF it is false. So, A implies B, B is false. So, definitely in this state's basically this formula this whole thing is not going to be true. It is something like A imply B and B is false A is true. So, therefore, this whole formula is not valid in state S 5 and S 2 and S 5. Importantly now let us think about S 1 and S 3 very interesting, here start is false here also start is false; that means, this one is going this two sorry this two basically is making this formula two vacuously.

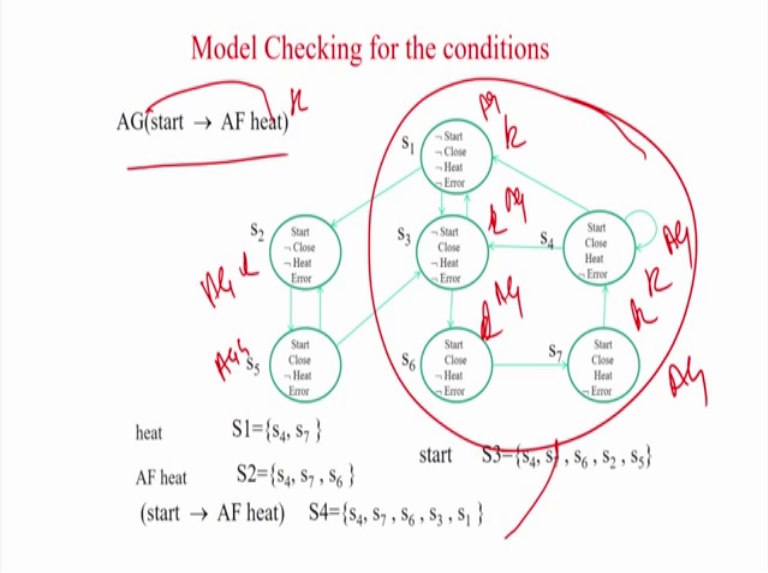
(Refer Slide Time: 64:57)



Because in this state's start is false. So, vacuously it is true.

So, A imply B. So, if A is false itself we do not we think that, the false system is true vacuously. So, we will have something like this like.

(Refer Slide Time: 65:11)



So, this is what is the steps. So, by labelling, so, it will find out that S 4, S 7, S 6 S 3 and S. So, these are the states where this is actually true now. So, what I have done? I have done it manually that is the vacuously true over here not this that, but in fact, basically you have to do it by model checking already. So, so already I have told you that for all

things like A imply B all other cases, there will be different ways of labelling and doing the model checking.

So in fact, we have shown only for the basic ones, but for all there be a model checking method algorithms will be there, what I have done manually the labelling will be done using an automated algorithm. But still now we are somewhere, where it says that start heat S 4 S 7 S 6, S 4 S 7 S 6 and S 3 and S 1. So, this is the set of states where this one is true now it says that all paths globing.

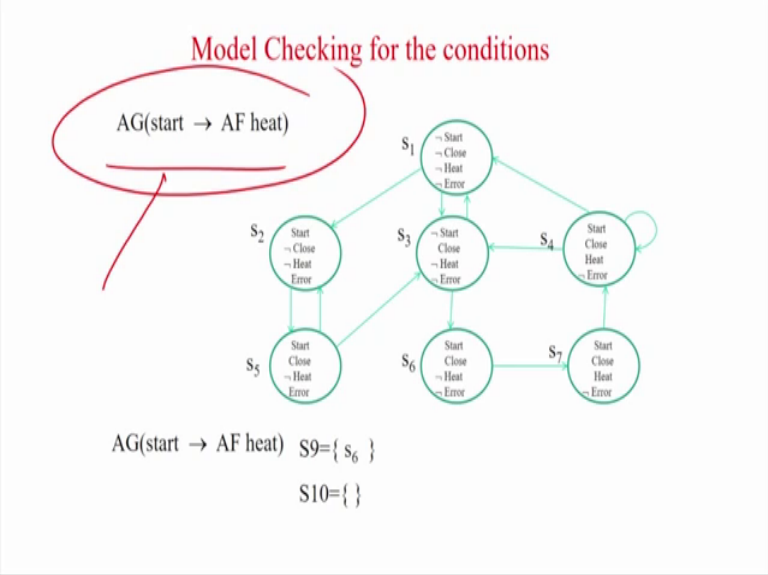
So, of course, if you look at it if you look at the last statement, what is says that here let us call it this whole thing let me call it as k. So, here this property is k is true over this k. Now it is saying all paths globally this should hold. Now already we have seen how to do that labelling. So, all the states will put all path globally and there will start eliminating states from which some path will be there, where it formula is false. So, everywhere will start putting AG k, AG k we will start writing AG k all this state's we will write even when this state also we will start writing right. Now sorry sorry ha it will be write, but then we will delete it, basically because k or this statement is not true over here of course, AG k it will be there, but then we will remove it. So, that will go over this states because there is there is some paths over here in which that it is not actually holding. So, that is by model checking algorithm AG k will be eliminated k is nothing, but this formula.

But now you see state S 1, S 1 there is at least a path now here where here in this case AG phi is not labelled. So, immediately this one also will start going of similarly if you look at it this one will be gone right. This state is gone no there will be another state here where this formula is not holding. So, again this one will go off right then basically if you look at it, now this state if you see there is one path over here where this k is not there a k can be eliminated. So, this labelling will also get removed.

Now, by virtue this has been removed. So, this one will also get removed similarly when this gets removed this one will get removed. So, slowly we will find out by steps that all these paths are gone and finally, you will end up in nothing. So, initially we will have this state's, but this states will also have the label, then this one will get this labelling AG label AG k, AG k will also be there they will be getting eliminated because k is false over here than this one will lose its label because this is not there then as S 1 will lose the

label, then S t will also lose the label similarly when S 3 will lose the label S 4 will lose the label, then S 7 then S 6 and finally, it will be a null set.

(Refer Slide Time: 68:22)



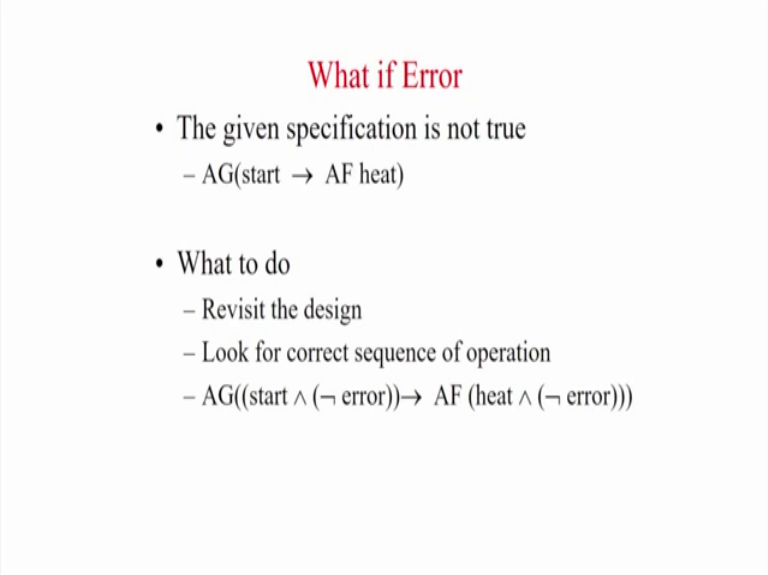
So, null set means none of this states will actually satisfy this formula. So, we have generated a counter example that what is the case start and in future heat why it is what is the problem. So, here it is very interesting. In this case this is this example actually has given you to modular system, which have shown how to modulus system how to write some temporal CTL formula that you want to verify, what it happens and then we have to see how we do model checking and then we find out states where it is violating.

So, now it says that the something has been violated now we have to debug what is the problem. First thing is that, whether this modelling is wrong that is one question. Secondly, whether this formula I have written is wrong. So, many things can go error of course, the model checking algorithm is assumed to be proper and it is known to be correct. So, there is no problem in the way we are doing the modelling, it is model checking.

So, the labelling and this properties are already known prove to be correct. So, that is no problem there. So, now, what to do? So, first I will check whether the modelling is proper or not, so in fact, the way we have discussed I you have to appreciate the fact there is no problem in the modelling the model is correct. Then what is the problem another way is that since somebody can tell me, that way I have written this specification

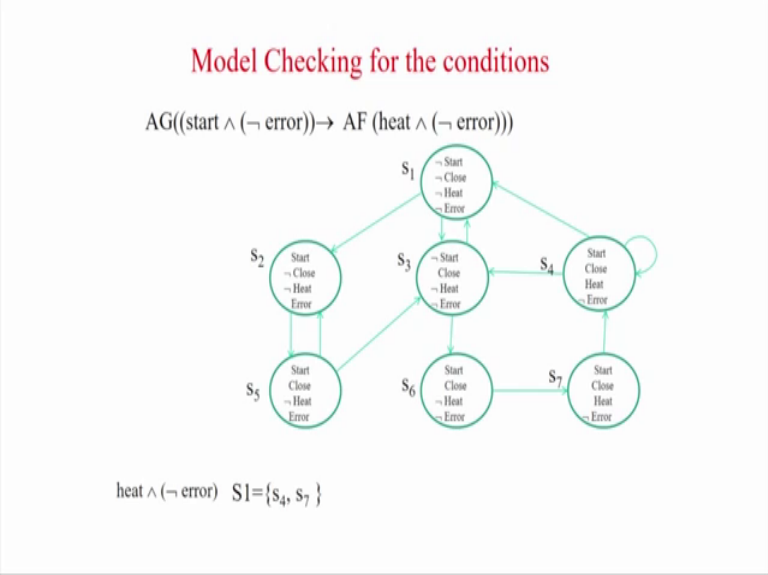
may also have an error. So, this example slightly is a twisted example because I could have also made model error and shown that how to correct it, but without doing that you can also try in your home, but there have try to given a different flavour here. So, what I have seen, I assuming that the property itself it in a slightly a problem.

(Refer Slide Time: 69:50)



What it should write this one, all paths globally start implies all paths in future heat, but not error that means.

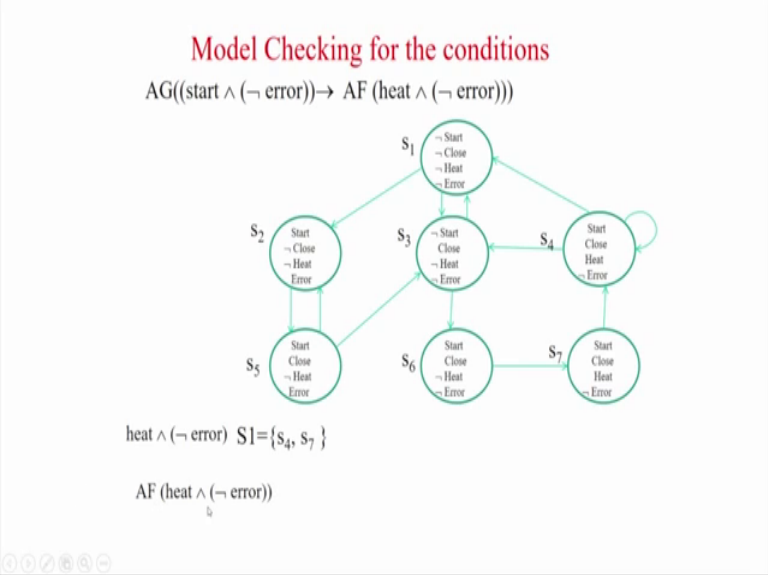
(Refer Slide Time: 69:59)



This is some path which was designed to have some error correction. So, if there is an error, then I do not bother about it because it a error case the heating is never on. So, if the error is on the heating is always off. So, that I always know. So, therefore, I have to think I have to or in other case, I have to always think that in the error is there I do not bother the property. The property says that in fact, start in future heat should be there, but it will hold if an only. If the error is not blinking if the error is blinking then basically you return will or micro will never start.

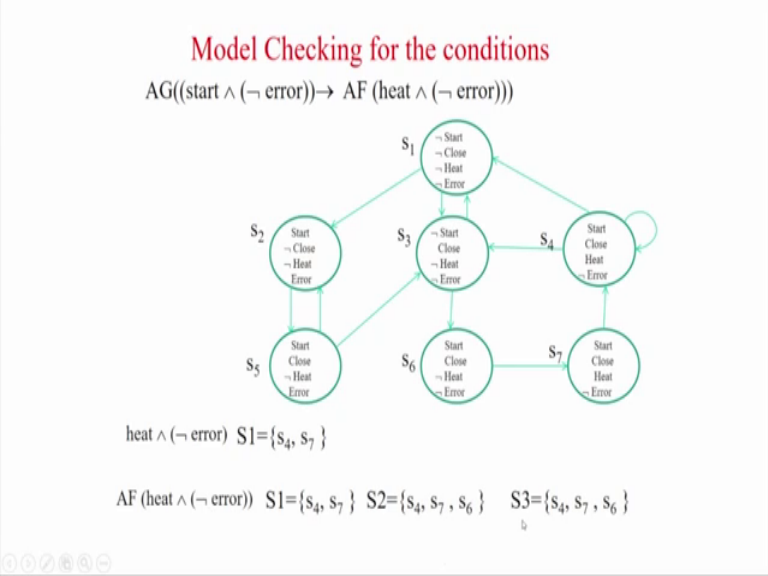
So, your correct property should be like something like start, and not error implies all paths that all paths in future heat and not error; that means, non error environment it should hold. Now basically I am not going to do the elaborately I discuss it, same thing you have to do only along with heat, it is first we are generating we started with this start so. In fact, now it is start and error.

(Refer Slide Time: 70:59)



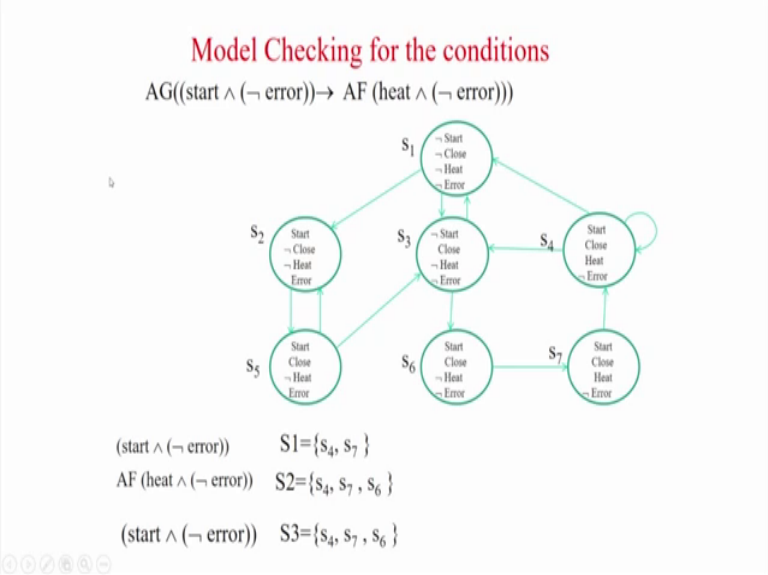
So, this states are S 4 and S 7 then all paths in future start and not error will be this state.

(Refer Slide Time: 71:04)



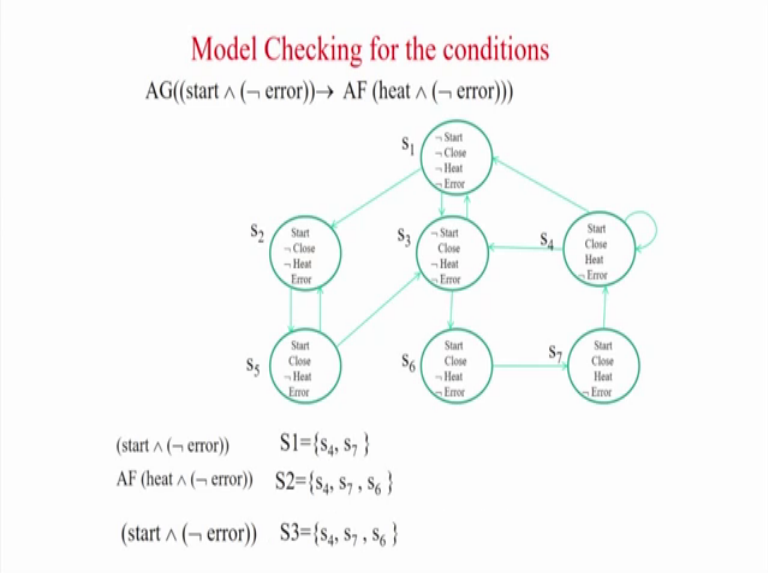
Finally it will land up to this stable state S 4, S 6 and S 7.

(Refer Slide Time: 71:07)



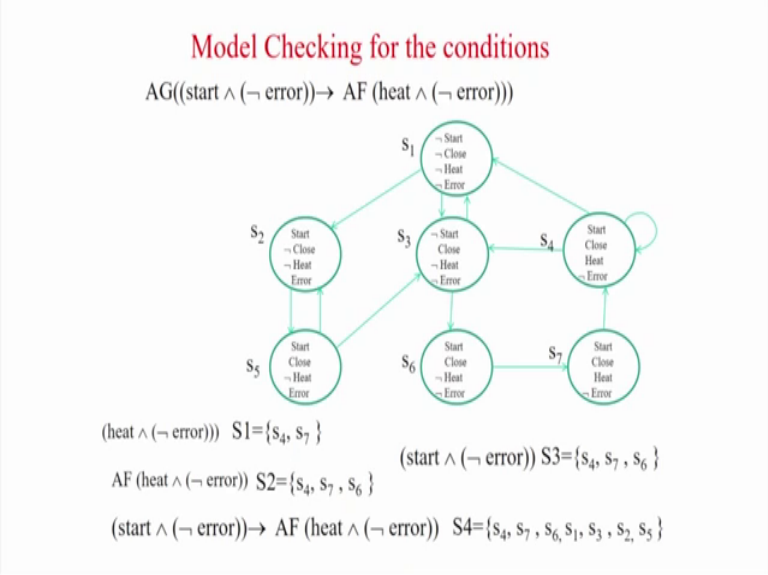
Then again you will have to go for all paths in heat similar way you have to find out the labels, for all paths in all paths in future heat and not label.

(Refer Slide Time: 71:15)



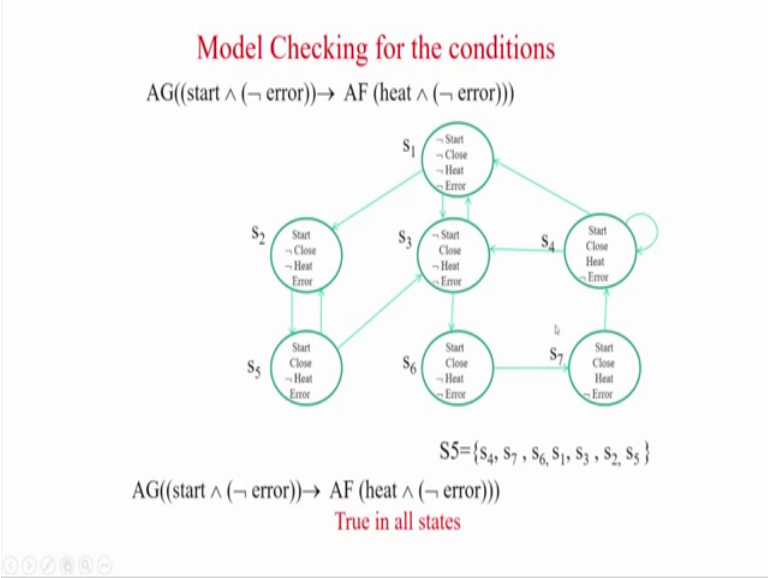
So, this is basically nothing, but your S 4, S 6 and S 7.

(Refer Slide Time: 71:18)



So, this way you are doing similarly you have to repeat for this one you know, this this state of states you know you check for implication.

(Refer Slide Time: 71:27)



And then finally, you will have you verify the property, as I have done you will find out that it will be satisfied in all the states, that I am just showing I have just shown in the slides I am not elaborately discussing, this is just a repetition just you look at this slide I am just holding this slide for slides will also be uploaded, but you can just see for the time being, here the states you do a manual calculation and you will find out that, it is to be in all the states.

So, therefore, in this case so, the stray example I had given, in which case I had made slightly errorless your basically specification. So, this has give you flavour that either there can be error in the specification either there can be error in the model, anyway it is very rare that both of them or any one of them has some errors, but still you are model checking means you are finding that all the properties are satisfiable.

Generally what happens, there is some errors in the model itself because is very complicated to build it specification are generally small way of representing what we want. So, generally there is no bugs here, but anywhere if anywhere there is a bug you will be give us error and then you have to find out which has to be rectified.

If the modelling has to be rectified, then there again can be two cases either the model a is a while you are translating the system to model there may be an error. So, then you have to modify the model itself or it may happen that, the system itself had some bugs

which is generally always want to catch, then you can go and tell guy see your this as a problem this account example you change.

So in fact, the in this formal model checking the advantage is everything is done mathematically. So, its are always exhaustive and guaranteed. So, generally if it says that the model is satisfying the properties, then it is guaranteed that for all input cases and those property models will be hit properly. So, that is the idea of formal verification over simulation based verification etcetera. So, this brings us to the end of this introductory lecture and what we have to take from here? From here you have to understand the basics of CTL model checking, there is another way of doing it called LTL model checking slight deviation that you can easily go home and read.

But what explicitly is issue the model. So, for a microwave there are 4 variables, you can find out that 2 to the power 4 16 states. For a very large system 100 and 1000 of variables you cannot even make the model itself. If you are not able to make the model itself forget just thing how can you verify. So, for to handle larger systems we will see how to optimise it how you use binary decision diagrams to do, it then also we will see high level abstraction and I will. So, many other different ways like model checking etcetera. So, this how will go in this module.

Thank you.