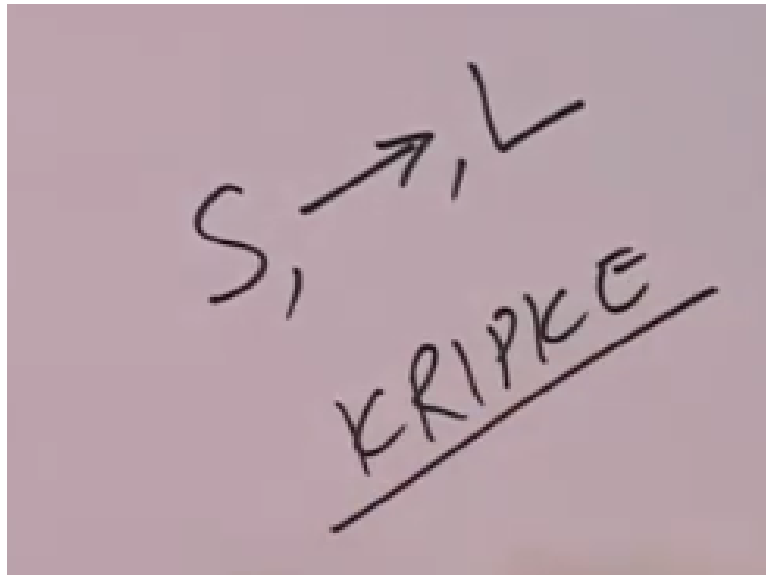


Advanced VLSI Design
Prof. Virendra K. Singh
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture – 42
VLSI Design Verification: Model Checking

Welcome to course on VLSI, advanced VLSI design. So, continuing with my earlier discussion, I will take you through the model checking techniques for design verification in today's lecture. So, in last lecture, we discussed that here if you want to verify VLSI circuit, then we need 2 things; one is the system model that we define as a transition system.

(Refer Slide Time: 00:54)



And we discussed that transition system is defined as a set of states, transition function and label on every state, that we obtained by converting the finite state machine to the Kripke structure. In Kripke structure, we transform the label from the edges in state transition diagram to the notes.

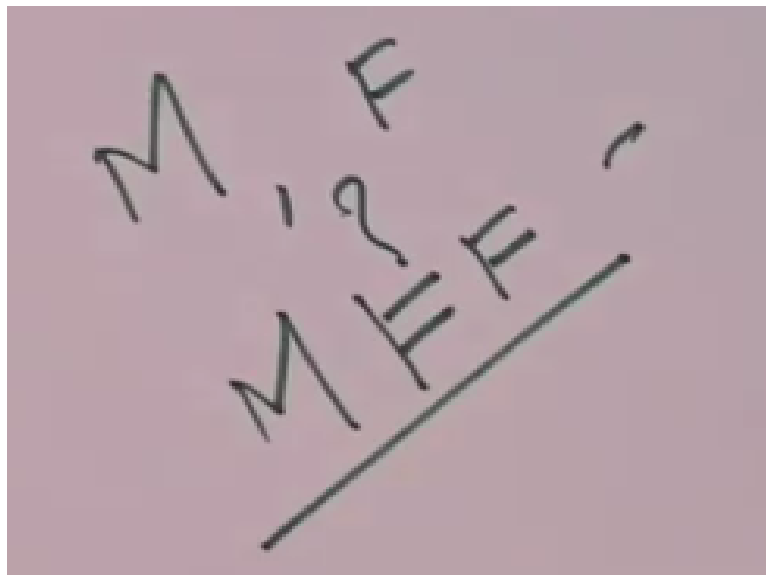
And the other thing that we need is; are the formula or the property that we want to verify and property is encoded in terms of temporal language and that is referred as temporal logic. So they, this is defined as formula in temporal language and we discussed that there can be; you can express in linear temporal logic or you can represent in the computation tree temporal logic.

(Refer Slide Time: 01:52)



So, LTL or CTL, I will take you briefly through both of this and then we will discuss this.

(Refer Slide Time: 02:06)



So, and now here, we want to say, we have model written as M, formula as F and then here we say that in some state, a model satisfies the formula F, if it satisfies in that case here, that property is all the time respected by your design. If it is not, in that case here it gives you counter example and that help you debugging the system. So, as I referred that we can use the linear temporal logic and then how formally, we define a formula in linear temporal logic.

(Refer Slide Time: 02:52)

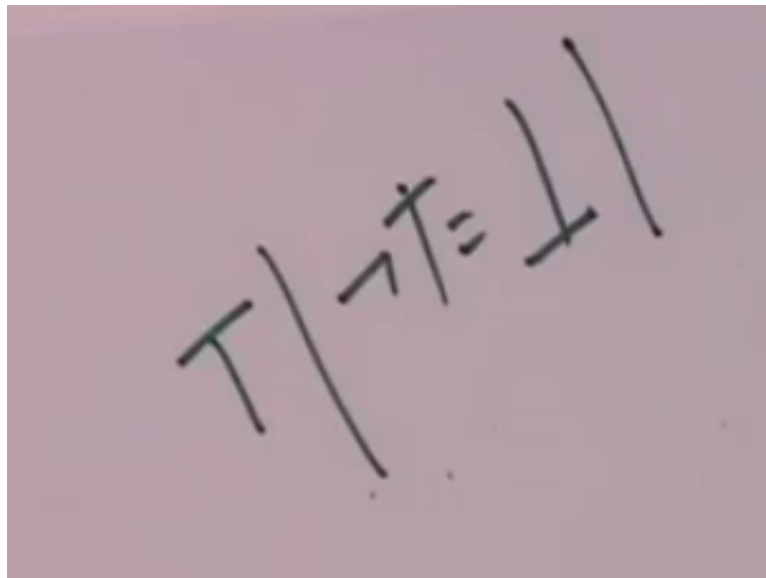
Syntax of LTL

Backus Naur Form

$$\phi ::= T \mid \perp \mid p \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \\ \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi U \phi) \mid (\phi W \phi) \mid (\phi R \phi)$$

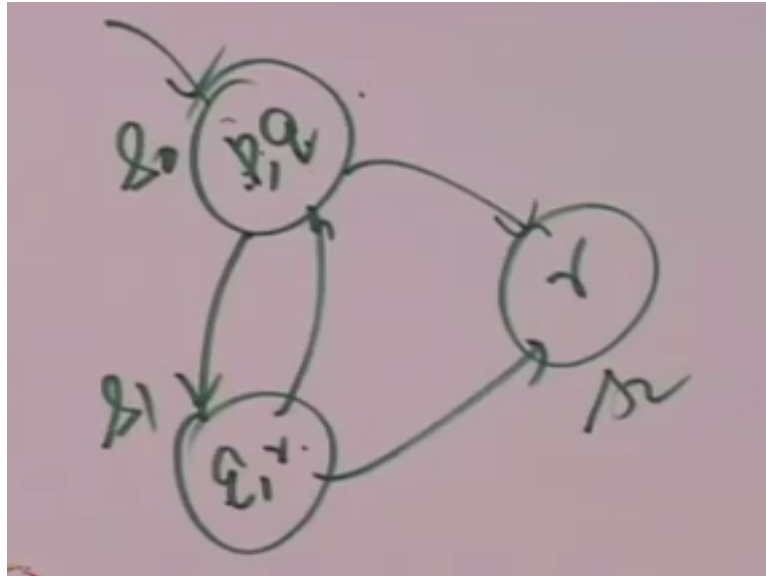
So, that formula can be defined as say, tautology, so that means your formula can be always true, that can be defined as say, sorry this is negation of tautology.

(Refer Slide Time: 03:06)



So this formula can be tautology or this can be negation of tautology or we can write like this or this can be any formula, which is labelled on some particular edge. So, when we convert our finite state machine into Kripke structure, we transform our input, output from the edges to the nodes and then nodes are labelled with the variables, which are true in that particular state.

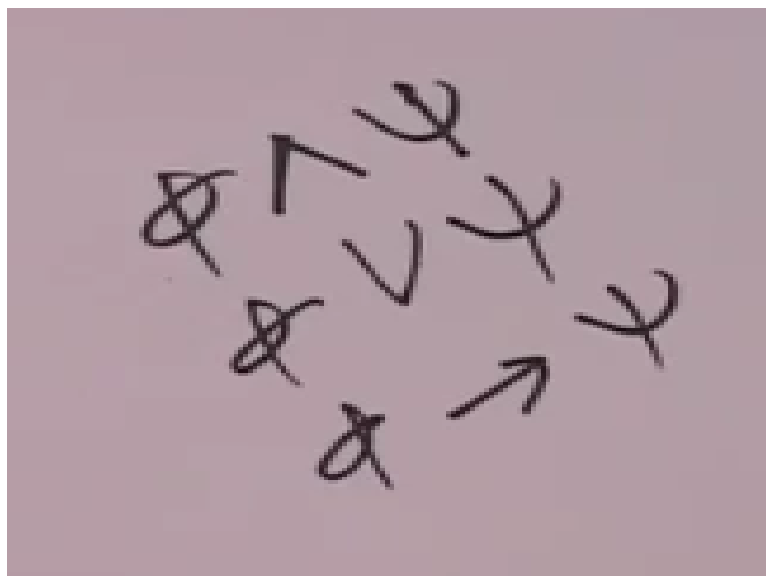
(Refer Slide Time: 03:46)



So, then all these variables, which are labelled on that particular state, are formulas. So, now here, the so a formula can be like here in some state transition diagram, if I say that this is my p and q are true here, q and r ; are true here, r is true in this particular state, say this is state s_0 , s_1 , s_2 and now here I can say this is my state transition; state transition diagram. Say, this is initially, state.

So, now here in this particular state, p and q both of the variables are true, so these are the true formula for the state s_0 . Here q and r both are true, that means these are true formula for state s_1 and r is true formula for state s_2 . Negation of any formula is also a, formula in LTL logic. There can be conjunction or there can be a disjunction of 2 formulas. So, now here you can say that in, there is one formula ϕ and another formula ψ .

(Refer Slide Time: 04:49)

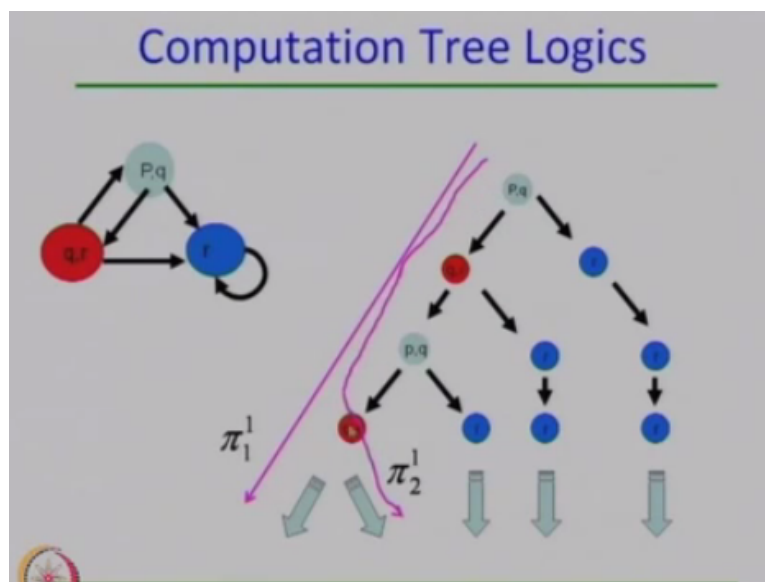


So, in that case here there can be say, phi conjunction psi and there can be 5 disjunction psi. There can be a 5 implication psi, so now here you can write in both of the; I mean all these ways. So, these are the static formula that normally we use in Boolean logic and I guess you understand the truth table of the static formulas. But now, in order to capture the dynamic behaviour of the system, here we also define some temporal operators.

And these temporal operators are X, F, G, U, W and R. X tells us the formula holds good in the next state, F tells you that formula hold good, sometimes in the future, the future may be next state, next to next state, next to next, next state and like this. Global operator tells you that this formula holds good in all these states in F path. There is a until operator; until operator says that one formula hold good until an other formula holds good.

In the same way, there is a weaker version of the until, that is called as weak until. So, that is defined as phi weak until phi, that mean, so this is; I will discuss how why this is weak and why we need that? And then here there is another operator that is called as release operator. So, these are 6 temporal operators; X, F, G, U, W and R. So, let us say this is a computation; this is your Kripke structure or a variant of finite state machine.

(Refer Slide Time: 06:57)



And now here, I unroll this in order to obtain a computation tree that how the computation progresses? So say, pq this is the state s0, that is the initial state and now if I start from this state, in that case here, this can either go to this state or it can go to this state, right. So, now here first time here, this will be or this will unroll like this. Now, when you come to this state in that case here, either you can go to this state or you can go back to the same previous state.

So, now here your computation tree can unroll like this. If you are in a state R in that case, you can be only in the state R. Now, in this way, you can generate the computation tree, which is infinitely long. So, this infinite long tree and it is half infinite width. So, this keeps on growing. Now, here any parts, is starting from the initial state or any progression is defined as a path. So, this is one of the paths.

So, and now here we defined this is the path 1 and which is starting from say state 1. Now, here there can be another path like here this one, so that is another path we define path 2 and then here this superscript tells you that from; which is the initial state of this path. So, now is path is starting from the state s1 and this is the second path. Now, here this I can say is third path starting from state s1.

(Refer Slide Time: 08:26)

Semantics of LTL

- A **transition system** $M = (S, \rightarrow, L)$ is a set of states S endowed with a transition relation \rightarrow (a binary relation on S), such that every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$, and a labeling function $L: S \rightarrow P$ (Atomic)
- A **path** in a model $M = (S, \rightarrow, L)$ is an infinite sequence of states s_1, s_2, s_3, \dots in S such that, for each $i \geq 1, s_i \rightarrow s_{i+1}$.
- Path is represented as $\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$
- $\pi^{s_1} = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$ (path starting from s_1)

So, now we define multiple paths in this computation tree, okay. So, as I am mentioned earlier that here; we model system as a transition system and we defined a path in this model, which is infinite sequence of the states s1, s2, s3 so and so forth and so a path is represented by phi and then here phi subscript s1 tells you that all the paths are starting from the state s1.

(Refer Slide Time: 08:56)

Semantics of LTL

Let $M = (S, \rightarrow, L)$ be a model and $\pi = s_1 \rightarrow s_2 \dots$ be a path in M . Whether π satisfies an LTL formula is defined by the satisfaction relation \models as follows:

$$1. \pi \models T$$

$$2. \pi \not\models \perp$$

$$3. \pi \models p, \text{ iff, } p \in L(s_1)$$

$$4. \pi \models \neg \phi, \text{ iff, } \pi \not\models \phi$$

$$5. \pi \models \phi_1 \wedge \phi_2, \text{ iff, } \pi \models \phi_1 \text{ and } \pi \models \phi_2$$

$$6. \pi \models \phi_1 \vee \phi_2, \text{ iff, } \pi \models \phi_1 \text{ or } \pi \models \phi_2$$

$$7. \pi \models \phi_1 \rightarrow \phi_2, \text{ iff, } \pi \models \phi_2 \text{ whenever, } \pi \models \phi_1$$

Now, let us look at the semantics of LTL, so now if there is a model M , that is a transition system and path π that is starting from s_1 , s_1, s_2 the infinitely long path in M , when π satisfies an LTL formula then this formula is defined as like here, tautology; that can be inverse of tautology. So, that means here it is always false or it can be a label on some particular state, so p and where p is the member of labels on state s_1 .

This can be inverse of any formula ϕ , so this holds good if ϕ does not hold good in that particular state and in a path. π can be the conjunction of 2 formulas; ϕ_1 and ϕ_2 , so this holds good and if and only if formula ϕ_1 holds good and formula ϕ_2 holds good or π can be disjunction, so that means here this π holds good. When ϕ_1 holds good or ϕ_2 holds good or there can be an implication.

(Refer Slide Time: 10:32)

Semantics of LTL

$$8. \pi \models X\phi, \text{ iff, } \pi^2 \models \phi$$

$$9. \pi \models G\phi, \text{ iff, } \forall i \geq 1, \pi^i \models \phi$$

$$10. \pi \models F\phi, \text{ iff, } \exists i \geq 1, s.t., \pi^i \models \phi$$

$$11. \pi \models \phi U \varphi, \text{ iff, } \exists i \geq 1, s.t., \pi^i \models \varphi,$$

$$\text{and } \forall j = 1, 2, \dots, i-1, \pi^j \models \phi$$

$$12. \pi \models \phi W \varphi, \text{ iff, either } \exists i \geq 1, s.t., \pi^i \models \varphi,$$

$$\text{and } \forall j = 1, 2, \dots, i-1, \pi^j \models \phi; \text{ or } \forall k \geq 1, \pi^k \models \phi$$

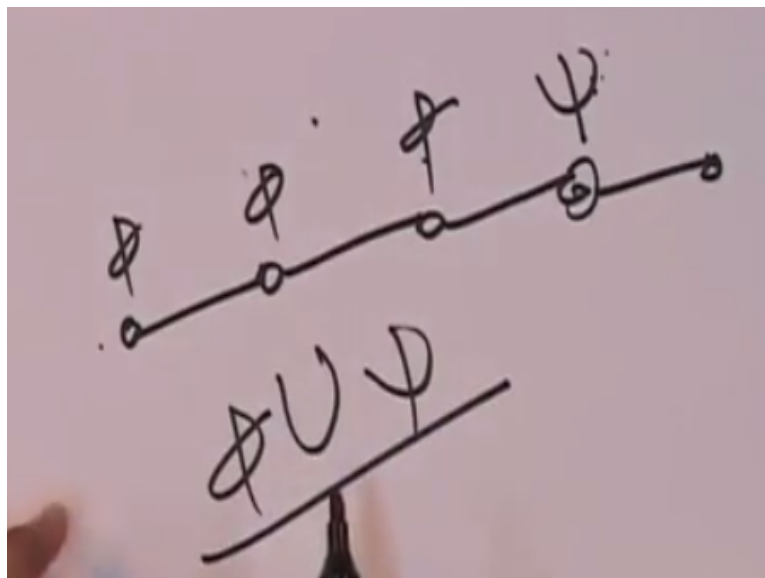
$$12. \pi \models \phi R \varphi, \text{ iff, either } \exists i \geq 1, s.t., \pi^i \models \varphi,$$

$$\text{and } \forall j = 1, 2, \dots, i, \pi^j \models \phi; \text{ or } \forall k \geq 1, \pi^k \models \phi$$

So, that means this says that ϕ_1 implies ϕ_2 . This means that, if ϕ_2 will hold good in that path, whenever ϕ_1 holds good. So, that means here we can reason about ϕ_2 only when ϕ_1 holds good, otherwise, we cannot reason about ϕ_2 whether it holds good or not. The temporal formula, here is defined as P_i satisfies X of ϕ if and only if. In the very next state formula ϕ holds good, G is a hold good if and only if in that path, everywhere ϕ holds good for i greater than or equal to one.

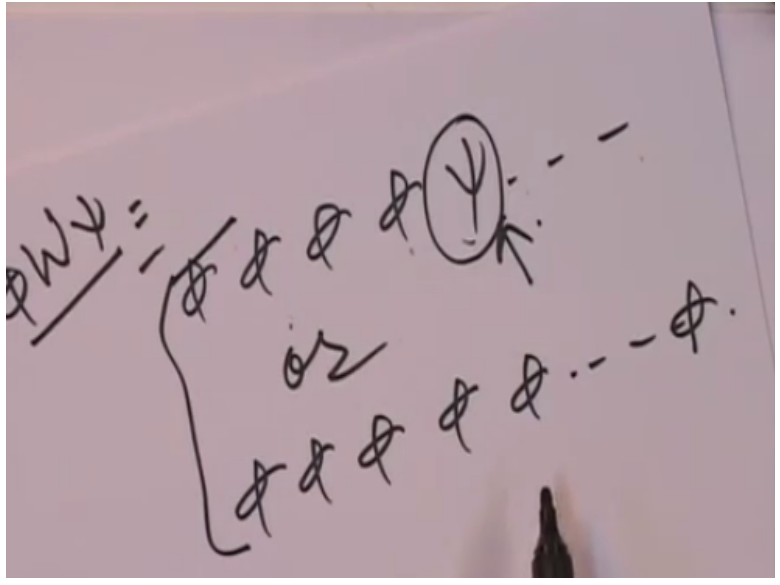
Future, somewhere there exist at least one state where in the formula holds good in that particular path, ϕ until ψ , ψ if and only if, for any value I greater than 1, if ϕ holds good, it means ψ holds good in ϕ , ϕ and for all value of j .

(Refer Slide Time: 10:32)



So, that means here, after particular state, so means this tells us that in the state progression, this ϕ holds good until ψ hold good, so until this point here, whenever you will get ψ until that point, ϕ should hold good. After that, ψ may hold good or may not hold good, so this is ϕ until ψ . This is a strong relationship, this says that ϕ should hold good until ψ holds good and ψ should hold good at least once in that particular path.

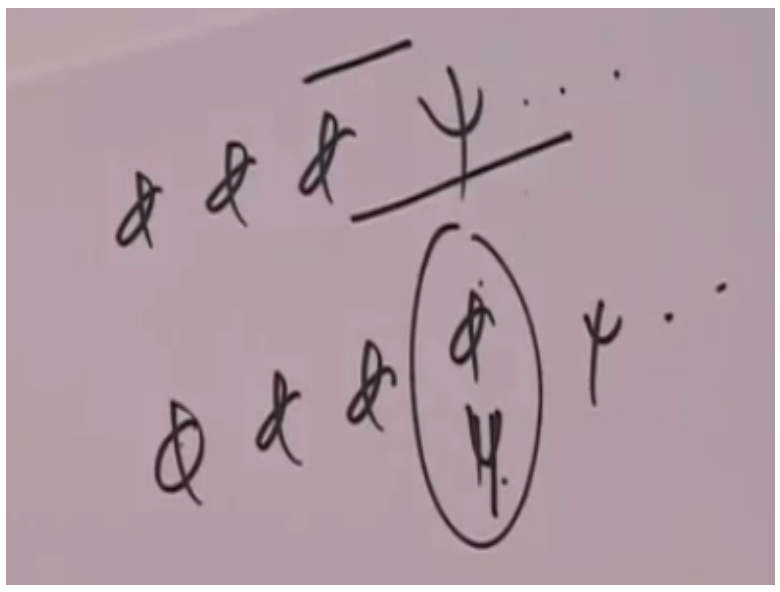
(Refer Slide Time: 12:16)



The W is a weaker version of that and that says that if strong until holds good, weak until also holds good, so that means here if you have this phi phi phi psi, then here phi weak until psi holds good or if so here, psi should hold good or if phi always hold good and psi never occur. In that case also here phi hold good, this phi weak until psi hold good. So, that means here phi phi phi infinitely long phi, this is also valid formula for phi weak until psi.

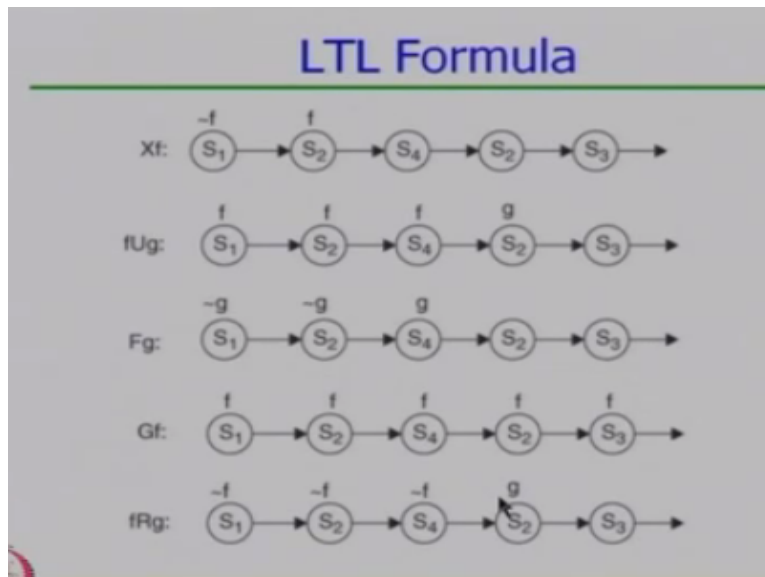
So, now here you can say that phi weak until psi is superset of phi until psi and in the same way here, you can say that there is another formula we say that phi releases psi, so that means here until this psi hold good, so that means psi should hold good until phi appears. So, in the weak until oh no sorry; in until and release, there is a relationship that in until, you need to have phi phi phi and psi.

(Refer Slide Time: 13:33)



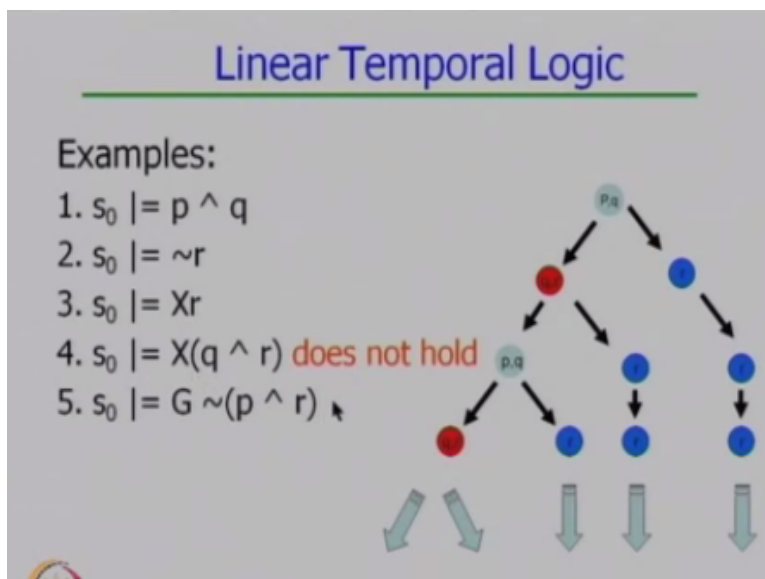
Whereas, in so and in the release operator you must have phi phi phi and at least in one state, you must have phi and psi hold good together, whereas that is not requirement in the until operator. So, these are the various operators we used to model, the system is to define the property of the system, keep in mind that here your implementation is modelled by a transition system and its specification is returned in LTL formula.

(Refer Slide Time: 14:14)



Now, we will take a; so these are some of the depictions of these formula as like here; X of f. So, like here F holds good in very next state, here it is false and here it become true, F you see is that, F holds good until g appears, F of g is the future operator so that means here, g holds good sometimes in the future. This may be after the 2 states, after 3 states or after 5 states. G of f is f holds good everywhere, f releases.

(Refer Slide Time: 14:55)



See is that here, this f should hold good until you, so now here you see in this state f and g both hold good. So, here now let us say, in this unrolled or computation tree, let us argue about these formula whether this formula hold good or not. So, now s_0 is p and q , so now here whether it holds good in the very first state that is s_0 , is holds good. Because p is true, so p holds good.

Q is true, so that means q hold good and then there AND operations are also holds good. So, now here this formula holds good. S_0 is negation of r , so that means here, r does not hold good, hence here negation of r holds good, so this formula is true, s_0 satisfies X of r , that means here in the very next state, r should hold good, so very next state from the s_0 state is either this qr state or rs state.

In both of the cases, r holds good, hence here you can say that this formula holds good. Now, if you look at X of q and r , so here in this state, q and r holds good, whereas in this state, q and r does not hold good hence means if this does not hold good in both of the next states, this formula does not hold good, so this formula does not hold. Like here, we can say, s_0 satisfies G of negation p and r .

So, p and r , so means both should satisfy in the paths starting from s_0 . So, now here in this p and r does not satisfy, here it does not satisfy, here it does not satisfy. So, in this path nowhere this satisfies ends, this formula is satisfied. So, now this way, we are here, we can write some of the formulas. So, now again here we have several operators in this. We have, means your conjunction and disjunction, negation and implication as static operator and X , F , G , U , W and R as temporal operators.

(Refer Slide Time: 17:07)

Linear Temporal Logic - Equivalence

1. $\neg (\varphi \wedge \psi) \equiv (\neg \varphi \vee \neg \psi)$

2. $\neg (\varphi \vee \psi) \equiv (\neg \varphi \wedge \neg \psi)$

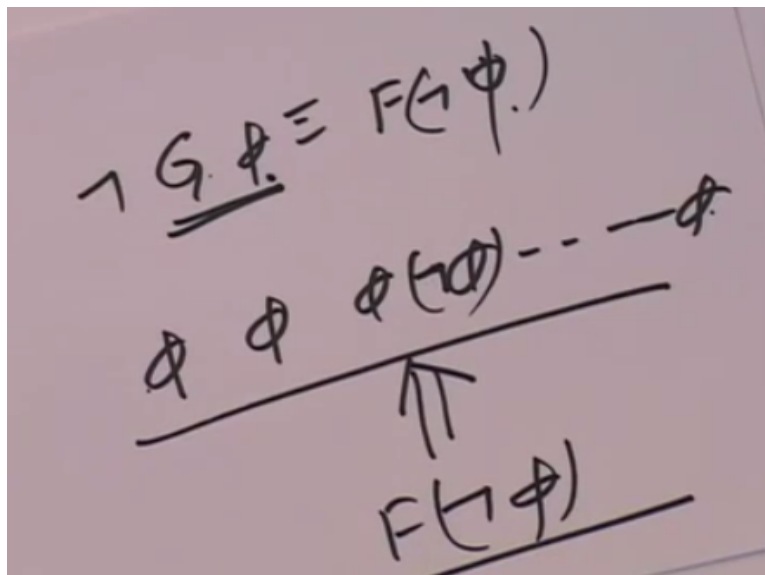
3. $\neg G \varphi \equiv F \neg \varphi$

4. $\neg F \varphi \equiv G \neg \varphi$

5. $\neg X \varphi \equiv X \neg \varphi$

So, now here, do we need all these operators like here if you want to define a system in that case here? Do we need negation conjunction disjunction all 3 operators? No, we do not need, do not need these operators and we know that there is a De Morgans theorem and then here De Morgans theorem can be defined like this one. I guess all of you know that, so I do not need to discuss this.

(Refer Slide Time: 18:04)

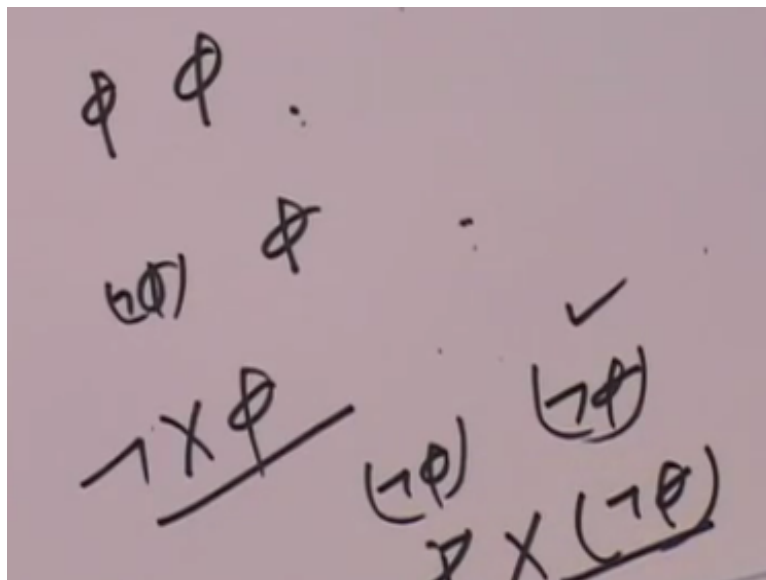


So here, now if this can give you the equivalence between different operators, this says that out of these 3 operators if we have 2 operators either negation and conjunction or negation and disjunction in that case here, these are sufficient to specify any of the system. Now, there is an equivalence relationship between temporal operators like here we can say, some of these operators are defined as like here you say, G of phi is equivalent to negation of G of phi is equivalent to F of negation of phi.

What this says is that, G of phi, negation of that, so that means here, when G of phi is satisfied when here whole along the path you have phi phi phi phi hold good. When this formula fails, this formula fails, if any of this state has negation of phi. So, if I have negation of phi here, in that case the negation of G of phi hold good, right. So, now this is the trace of negation of phi.

Now, in this what I can say is that, somewhere in the future at least in one of the states, negation of phi holds good. So that means here, I can write this formula as F of negation of phi. So, that gives you the logical equivalence of negation of G of phi is equivalent to F of negation of phi. This way you mean, you can write this other way round as negation of F of phi as G of negation of phi.

(Refer Slide Time: 19:24)



In the same way, X of phi, so now in this state, is you have phi; next state, you have phi. So, say the current state, you may have anything, but phi or negation of phi. But in the next state, if you have phi in that case your formula, phi holds good. Whereas, if you say that this formula is does not hold good or does not satisfy in that case, your trace must be here you may have negation of phi and the next state must be negation of phi.

(Refer Slide Time: 20:11)

Linear Temporal Logic - Equivalence

1. $\neg (\varphi U \psi) \equiv (\neg \varphi R \neg \psi)$
2. $\neg (\varphi R \psi) \equiv (\neg \varphi U \neg \psi)$
3. $F\varphi \equiv T U \varphi$
4. $G\varphi \equiv \neg T R \neg \varphi$
5. $\varphi U \psi \equiv \varphi W \psi \wedge F\psi$

This is equivalent to X of negation of phi because here in the next state, negation of phi holds good. So, this way we can write equivalence. In the same way, you can work out there is an equivalence relationship between you and operator, so means negation of phi until psi is equal to negation of phi until negation of psi. This relationship is like De Morgans theorem or other way round you can also write it.

(Refer Slide Time: 20:36)

$$F\varphi \equiv (T)U\varphi$$

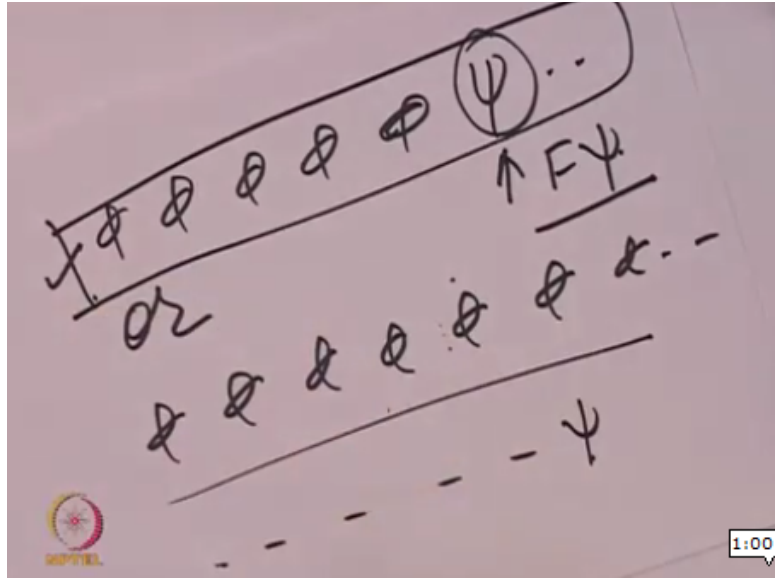
$$\neg\varphi \cdot \neg\varphi \cdot \neg\varphi \cdot \varphi$$

$$\downarrow$$

$$\underline{F\varphi}$$

In the same way here, you can write, F of phi is equivalent to tautology until phi. Tautology means here no restriction always holds good, right. So, that means here, without any restriction in some state, here phi holds good, so that means here you have negation of phi, negation of phi, negation of phi and in some state, phi hold good. So, this is equivalent to what? This is equivalent to F of phi because somewhere in the future your phi holds good.

(Refer Slide Time: 21:41)



So, this way you can write like this. In the same way, here global operator, you can write $G\phi$ is equivalent to negation of tautology, releases ϕ , so the negation of tautology and as I explained you earlier that your weaker until is super set of until, so that means here, if I had something in; so now your ϕ until ψ is defined as $\phi\phi\phi\phi$ and somewhere ψ , right. Weak until can be either this one or $\phi\phi\phi\phi\phi\phi$ everywhere ϕ , right.

So, if this is there in that case, this is equivalent to weak until. Otherwise, I have to have somewhere ψ should hold good. So, this I can also write as in order to capture this one, I can also write as this as $F\psi$, $F\psi$ means here ψ hold good somewhere in the future. So, your strong until is equivalent to weak until and $F\psi$, that gives you exactly this thing. So, this way you can have equivalence between the weak until and strong until.

(Refer Slide Time: 22:42)

Linear Temporal Logic - Equivalence

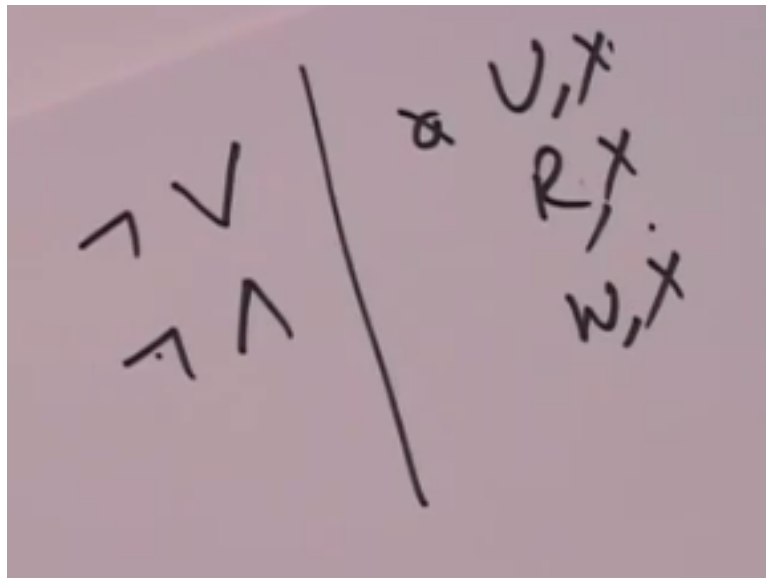
1. $\phi W \psi \equiv \phi U \psi \vee G \phi$
2. $\phi W \psi \equiv \psi R (\psi \vee \phi)$
3. $\phi R \psi \equiv \psi W (\psi \wedge \phi)$

Essential Set

- ✓ {U, X}
- ✓ {R, X}
- ✓ {W, X}

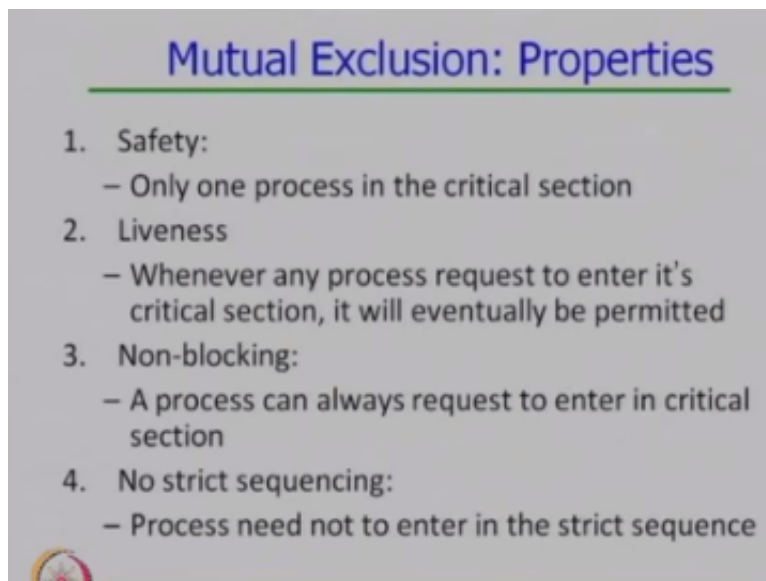
So, now here in this way you can define the equivalence between these operators. Now here, if you want to see, which are the essential sets in that case here out of these 6 operators, you can say, I need either U or X or R or X or W or and X. So, now either set is okay, so now here you need 2 operators from the temporal logic and 2 operators from the static logic and so now here either from the static if you need to have either this set or you need this set.

(Refer Slide Time: 23:18)



Both of this sets are okay and from the temporal, you need to have UX, this one or this one, so now here you can pick any of these 2 and any of these 3, so and that can be defined you the complete set.

(Refer Slide Time: 23:43)



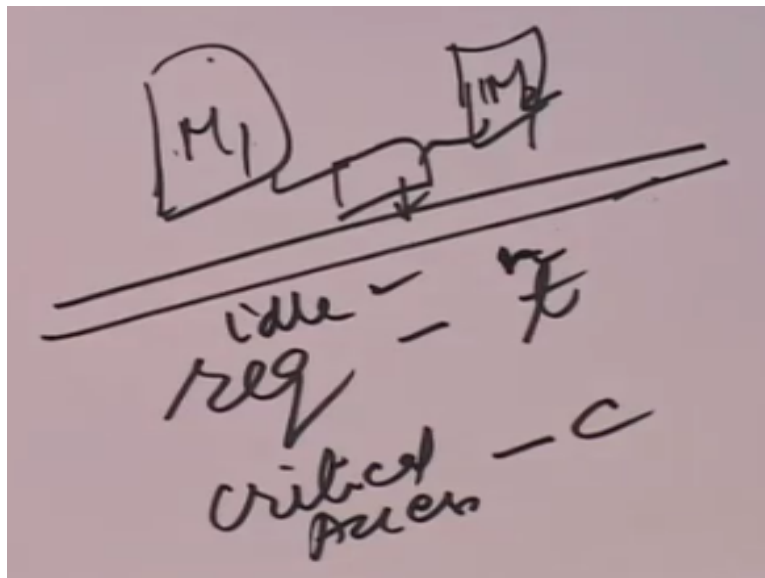
So, now here the question is what we can do with this? How we can write or how we can specify property of the system? Let us take a simple example, this example tells you the

mutual exclusion or arbiter that gives you access to the common resource. What we want from that? There are couple of things that we need from this system and those are defined as like here, if one; master one to get access to the common resource.

In that case you are; it should be allowed to get access. Two masters cannot get access to the common resource, right. So, and now here it should not be blocking, so that means here like whenever somebody wants to access, sometimes in the future, it should give access to the common resource and it should not allow the strict sequencing that means here it should not, means say give you the say, Round Robin policy or something like that.

So that first gives the access to the master one, then master2, master3, master4.

(Refer Slide Time: 25:00)



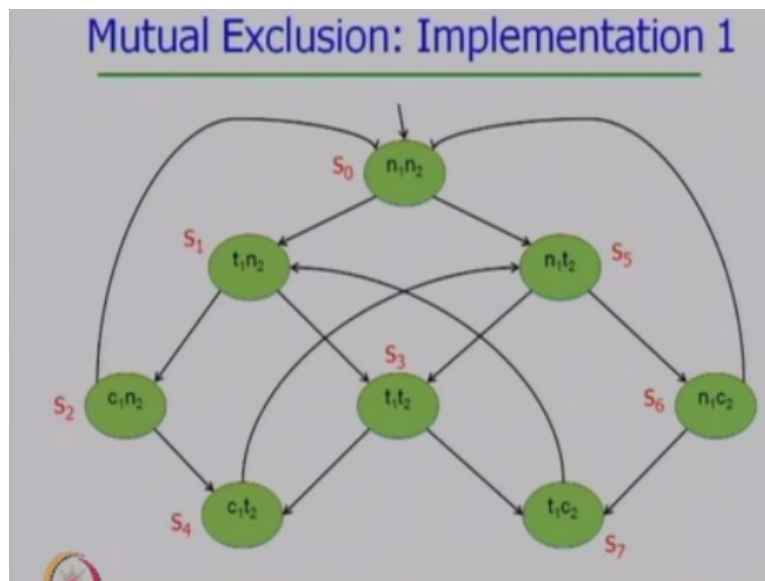
So, here you have 2 masters, say M1 and M2 and say they want to get access to the common resource and this common resource may be the boss. So, now here this master can be in any of the state, one state may be here it may request. I say that is trying state, that is, it is trying to get access and then here ones it gets the; your request is registered, you get access to the common resource and you are allowed to transfer your data to the common boss.

And that state here; we say as critical state. So, now here you are in the critical state and you have access to the system say that is state c and now here many times your master may not want to have the access to the common resource, so then you can say that it is not trying for anything it is in the idle state. Idle state, I say referred as M. Now, what are the important properties we are interested in?

One of the property is that is defined as safety property and safety property says that only one process in the critical, only one process should be in the critical section. So that means, here more than one process should not access to the common resource. Another property is defined as liveness and liveness says that whenever any process request to enter in the critical section, it will eventually be permitted.

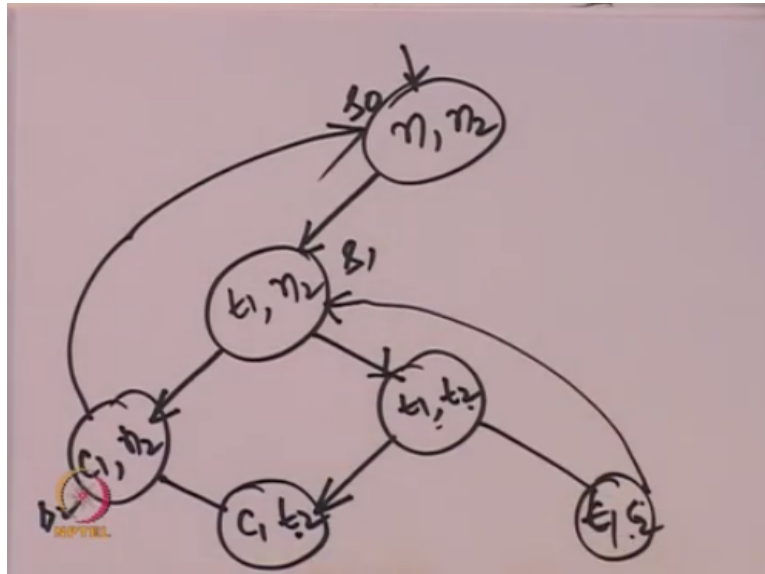
So, that means it is not necessary that in very next state, it should be permitted but it should be permitted somewhere in the future and third property is non-blocking property, we are interested in and this says that a process can always request to enter in the critical section, once it gets access to the critical section, it should not be disallowed for that, it can always make any request at any point in time and fourth property that is very important is, no strict sequencing.

(Refer Slide Time: 27:45)



You may or means some of the protocol you may use Round Robin and then this property is violated but ideally you may want that here. There should not be a strict sequencing, so that means here, process need not to enter in a strict sequence that here first process p1, then p2, p3, p4 something like that, it should not happen like that. Okay, so now here we design a mutual exclusive system and that system will work like this.

(Refer Slide Time: 28:01)



So, let us work out it little bit. So, say initially both of the resources or masters are not requesting for any resource, so they are in state, where in n_1 and n_2 , n_1 says that process p_1 does not request process and to say process p_2 does not request. Now, so say this state is s_0 , now when you will get request from master M_1 then it will go to the state M_2 and now sorry master 1 or process p_1 , then it should go to the another state and in that state say t_1 is in trying and n_2 is not trying, right.

So, this is in the trying state that means it rays the request after that what will happen? Now, in this either your t_1 or process p_1 will get access, so if it gets access in that case here, it will go to a state c_1 and t_2 is means the process p_2 is not trying or process p_2 may also start to try. So, now here it will start to try and then here, it will go to a state t_1 and t_2 . So, let us say come back to process p_1 , which gets the access so this was s_1 , this is s_2 .

Now, once it completes here, it goes back to the initial state, right. Now, here or from this state, if t_2 start to; t_2 again reject the request, so far there was no request. So, in that case here it will go to state c_1 and t_2 . Come back to the previous state t_1, t_2 when it is in t_1, t_2 , in that case here, either it will give access to; I mean process p_1 or it will access to the process p_2 . So, say it gives access to the process p_1 .

So, this will work like this or it gives the access to the process p_2 so in that case here, this will give you $t_1 c_2$, right. Now, if it goes to $c_1 t_2$ in that or it goes to $t_1 c_2$, in that case here, once it completes, t_2 completes here, it comes back to the state t_1 and to because now here

process second is not requesting for anything. So, now here like this, you will also get the similar kind of state graph this side.

So, now here if you look at this, you will get a state transition diagram like this which gives you the implementation of an arbiter or mutual exclusion. I hope this is clear to you by this time. Now let us see how I can encode these properties and for this one and check whether this implementation is correct or not. So this is implementation and this is Kripke structure. So, Kripke case structure tells you that here these variables t1 and n2 are true in this state.

Here, n1 and n2 are true in this state and in this we have 6 different variables t1 t2 n1 n2 c1 c2, right. Now, here let us look at the properties. So, first property was the safety property and safety property says that only one process should be in the critical section, how I write that? So that means in every state in this diagram, I have to say that only one process should be in the critical section.

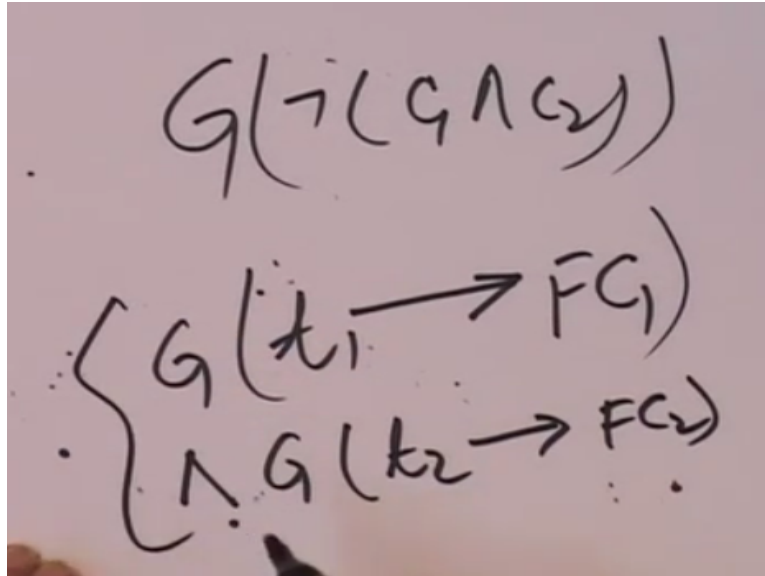
(Refer Slide Time: 32:05)

Mutual Exclusion: Properties

- Safety**
 - Only one process in the critical section ✓ **Pass**
 - $((G \neg(c_1 \wedge c_2))$
- Liveness**
 - Whenever any process request to enter its critical section, it will eventually be permitted **X Fail**
 - $(G(t_1 \rightarrow Fc_1))$ Counter Example: $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \dots$
- Non Blocking**
 - For every state satisfying n_1 , there is a successor satisfying t_1
 - Not expressible in LTL
- No strict sequencing**
 - $G(c_1 \rightarrow c_1 W (\neg c_1 \wedge \neg c_1 W c_2))$ ✓ **Pass**

So, that means here either c1 should hold good in any of the state or c2 should hold good, both should not hold good, right. So, means that how I can define? I can define that globally c1 and c2 both should not hold good.

(Refer Slide Time: 32:13)



So, that means here c_1 c_2 should not hold good that means here the negation of this should hold good all the time globally. So, I can write this property like this. Second property is liveness property, which says that whenever any process request to enter in to the critical section, it will eventually be permitted, so that means here, whenever a process wants to enter, so that means whenever it is trying in that case here, somewhere in the future, it should get access.

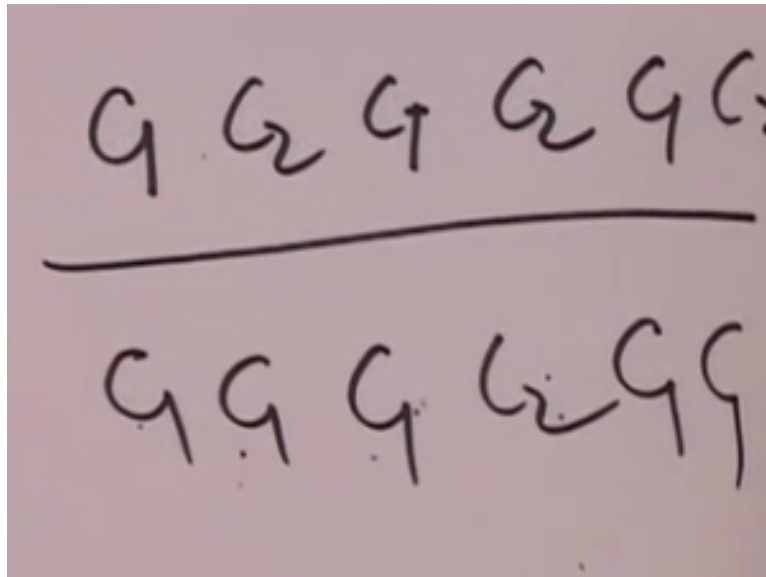
So, that means here now whenever some process say, if t_1 is trying, so in that case, after several cycles I do not know how many cycle? But after several cycle, it should get access so that means here it should go in the critical X section or it can get access to the common resource. So, now here this should hold good and this should hold good globally. This is for process p_1 , this should also hold good for process p_2 , so that means here t_2 implies F of c_2 . So, now here this encodes liveness property. Third property is non-blocking property.

So, non-blocking property say that for every state satisfying n_1 , there is a successor state t_1 , so that means here it should not be at least in some of the parts, the next state should be after n_1 , it should be t_1 , right. But now here in this we cannot select or define property for selective parts. Here this property says that here this need not to hold good all the time, this should holds good selectively.

So, in that case here, we have to; we cannot express that and I will come back to this point and will tell you that this can be express in computation tree logics; computation tree temporal logics, CTL and this is not expressible in LTL. The another property is, no strict

sequencing, no strict sequencing says that you cannot restrict sequence; restrict sequence tells you that you have to go in this way.

(Refer Slide Time: 34:37)

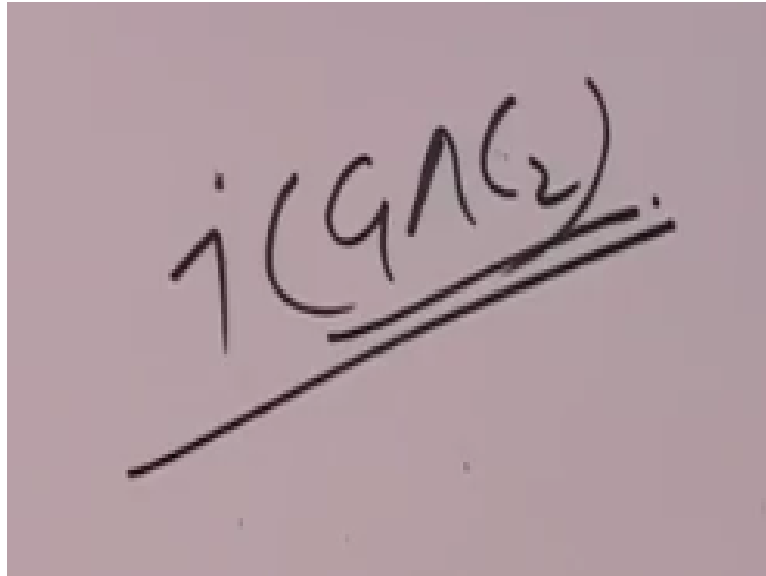


First c_1 goes in the critical section then c_2 , then again c_1 , then c_2 , then c_1 , c_2 it has to follow the strict sequence and what we want is that? it should not have this strict sequence. So, that means here this may follow the sequence as well but at least there must be one path in which it should go like $c_1, c_1, c_1, c_2, c_1, c_1$ something like that. So, that means we have to; now here, how I can write this property?

We can write this property as a negation of the property because here again, we cannot write property for the selective formula, but in this particular case, negation can help us that if we say that there is at least one path in which this strict sequencing that is, c_1, c_2, c_1, c_2 does not hold good that means here it does not follow a strict sequencing in all the time. So, in this case, we are; we can write this property as a negation of the property.

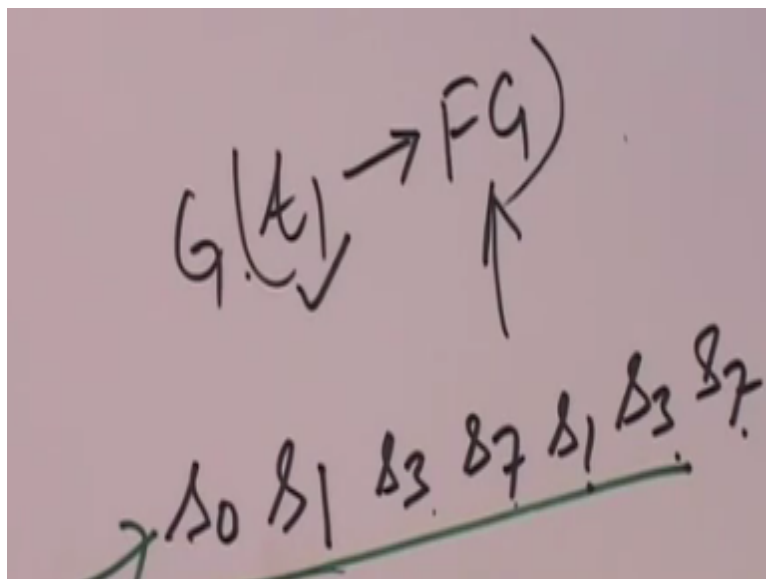
And that says that here your c_1 either should hold good all the time and so that means here c_1 implies c_1 , weak until, so that means here either c_1 continues or it should go to the negation of c_1 and then here from negation of c_1 , it goes to c_2 and then come back to c_1 . So, now here this says that here is strict sequencing is not needed and if this property fails that means here a strict sequencing passed, no strict sequencing passes.

(Refer Slide Time: 36:38)



So, this way we can write. Now, in this state transition diagram, if we look at this property, so if you go back to the state transition diagram here, you can say that nowhere in this c_1 and c_1 both are true, so if this false, so that means here negation of this is true. So, that means here the first property holds good. Second property tells you that what it says?

(Refer Slide Time: 36:55)



That you have to have $t_1 Fc_1$ and this holds good globally. So that means here after t_1 , somewhere in the future you should get c_1 , right. Now, here if in this transition diagram, if you look at a path $s_0 s_1 s_3 s_7$, you are what you are getting? Here $n_1 n_2$ here you will get t_1 , so now here t_1 is true. Now, you can reason about Fc_1 , once t_1 holds good. So, t_1 holds good here. Now, after that what we want? That somewhere in the future, it should; c_1 should hold good.

So, that means from t1, if I look at this one, t1 here you will get c1, so in this path, this holds good, but if you look at this path s0 s1 s3 s7 s1 s2 s3 s7 s1 s3 s7, so the path that is s0 s1 s3 s7 s1 s3 s7 and so on and so forth. In this path, here you have t1 t1 t1, again t1 t1 t1 t1 and so you will never get c1. So, that means this path, while it is this property. So, what it says that your second property, which is liveness property fails.

So, that means, and what liveness property says? That whenever any process request to enter into the critical section, it will eventually be permitted and here it means this counter example that we have discovered s0 s1 s3 s7 s1 s3 s7, so if it follows this computation path, it would not permit t1 or the process one to enter into the critical section. Hence there is a design bug and it captures the design bug.

I will come back and tell you how we can look at, means how we can debug that design bug? Non-blocking, so the counter example tells you that here, which path violates this property. Now, third property is not anyway not encoded in LTL, then go back to the fourth property and then here you will find out; find in the computation path you can do this on your own that this is always; this property is always is respected.

So, that means your negation of this property holds good. So, now here how I can say this process right. If you look at this, where is the source of error? Source of error lies here at s3 when you come to state s3, where in t1 and t2 both are true, now you lose track through which path you came from s0 s1 s3 or you came from s0 s5 s3 and now here this can go to this state or this can go to this state.

So, now here, you do not have memory, otherwise if you remember that if you come through this path then here you would not allow it to go here you can allow it go here. So, now means in order to debug it or in order to fix this bug say, I mean say for debug, we get to know that this is the source of bug and in order to fix this bug here now what is the solution, what is the remedy?

Remedy is that here you split this state in 2 states, so that you can remember that whether you came through this path or you came through this path and so now here that will fix your problem. So, now here by splitting this state in state s3 and s9, both have t1 and t2 variable

high or a true, so in that; but here now, I remember if it came through this path in that case here, this would go here.

(Refer Slide Time: 41:37)

Mutual Exclusion: Properties

1. Safety
 - Only one process in the critical section ✓ Pass
 - $((G \neg(c_1 \wedge c_2))$
2. Liveness
 - Whenever any process request to enter its critical section, it will eventually be permitted ✓ Pass
 - $(G (t_1 \rightarrow Fc_1))$
3. Non Blocking
 - For every state satisfying n_1 , there is a successor satisfying t_1
 - Not expressible in LTL
4. No strict sequencing
 - $G(c_1 \rightarrow c_1 W (\neg c_1 \wedge \neg c_1 W c_2))$ ✓ Pass

Advance VLSI Design 17

If it came through this path, it should go over here and now here there is no confusion and now if you look at it you would not find such a path and hence here both all these 3 properties will pass and so now here this way you can specify property and then you can verify those properties on a given model and so now here that model is generated from the implementation.

So this transition; implementation is describe the transition system or finite state machine or Kripke case structure and your specifications are returned in terms of LTL formulas.

(Refer Slide Time: 42:12)

Linear-time Temporal Logic

- LTL formulas are evaluated on paths
- State of a system satisfies an LTL formula if all the paths from the given state satisfy it
- LTL implicitly quantify universally over paths
- Properties which asserts the existence of a path cannot be expressed in LTL
 - Negation can partly solve the problem
 - Check whether all paths satisfy the negation of formula

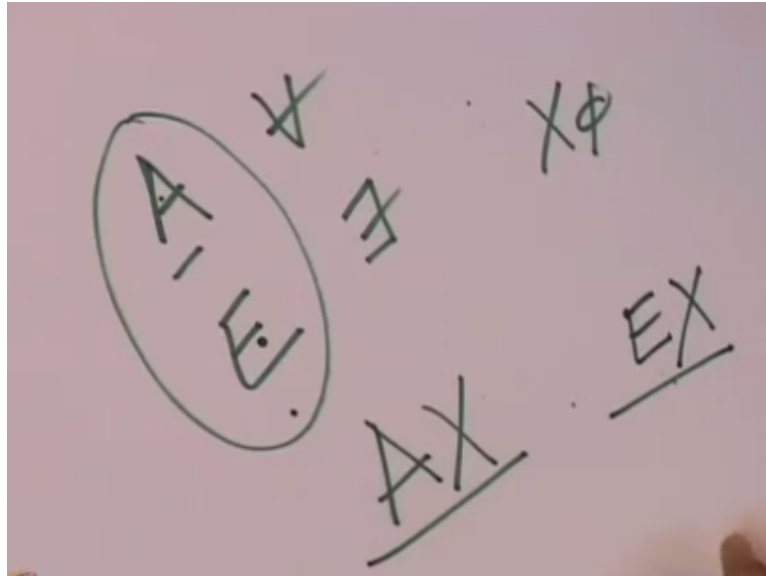
And, as we know that here LTL formulas are evaluated on paths, so that means here and that is why this is linear because on a path always with a clock tick here, you were linearly move to the next state, next to next state, next to next state always. State of the system satisfy LTL formula, if all the paths on a given states satisfies the formula. So, that means from one state if n number of paths are originating in this case here that state satisfies formula only when all paths originating from that states satisfies that formula.

Now, here LTL implicitly quantifies over the universally over the paths, so that means here, all the paths originating from the semi state and now here, but some of the properties, which asserts the existence of path, so that means those are defined for a selected set of path cannot be expressed in LTL, that is; this coming out of LTL. Sometime like here as we have seen that no strict sequencing, we can use negation and we can defined the property.

So, negation can partly help us but not all the time. So, for that here we need to go to the branching type temporal logic. So, if you look at the computation tree, in that case from every state here now these branches out either it goes to this state or it goes to this state. So, now if you start to define formula based on the computation tree and or you can say the; with respect to a state, in that case here, we can select some of the paths.

So, now here from either it goes here or it goes here, so now here we can select the path and so now that formula that is defined as computation tree temporal logic CTL. So, now here a big difference between the LTL and CTL is that with every formula, or temporal operator, we need to specify one path modality, whether this holds good on all the parts originating from a state or it holds good on selected path.

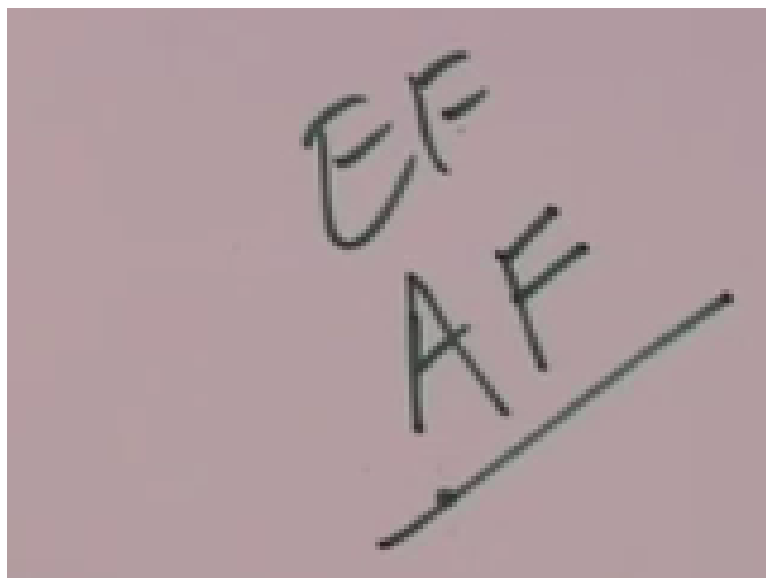
(Refer Slide Time: 44:42)



So, one path modality is defined as A, that is for the representing the universal quantification, so that means it depend it define for all the paths originating from one of the state or we define E that refer the existential quantifier that means here, it says that this exist or this holds good at least for one of the paths and so now means, we have to associate these path modalities with each and every temporal logic.

So, now here your static formula remains same, but the temporal formula has association with the path modalities either universal or existential. So, now if you say X operator I have that says that some formula holds good in the next state. So, now there can be several next states from one particular state. So, now here based on whether it should hold good in all the next states or it should hold good in one of the next; at least one of the next states.

(Refer Slide Time: 46:24)



We defined the path modality, so it should hold good in hold the next state in that case we defined this; we associate A or if it should hold good on selected or at least one of the next state in that case here, we defined this as E of X. So, in the same way here, we can write; we can define if some formula holds good, some times in the future in any of the path in that case here, we define that as E of F or if it holds good in all the paths in that case we are defined as A of F.

So, now here means, we can define these as tautology or sorry there must be negation here. So, this is negation of tautology or this can be a formula which is label on particular state, the CTL formula can be negation of a formula, it can be conjunction, disjunction, implication, it can be temporal operator X and that is associated with path modality as A, it can be associated with path modality as E.

(Refer Slide Time: 47:31)

Semantics of CTL

Let $M = (S, \rightarrow, L)$ be a model for CTL, and s in S , ϕ a CTL formula. The relation $M, s \models \phi$ is defined by structural induction on ϕ .

- 1 $M, s \models T$
- 2 $M, s \not\models \perp$
- 3 $M, s \models p$, iff, $p \in L(s)$
- 4 $M, s \models \neg \phi$, iff, $M, s \not\models \phi$
- 5 $M, s \models \phi_1 \wedge \phi_2$, iff, $M, s \models \phi_1$ and, $M, s \models \phi_2$
- 6 $M, s \models \phi_1 \vee \phi_2$, iff, $M, s \models \phi_1$ or, $M, s \models \phi_2$
- 7 $M, s \models \phi_1 \rightarrow \phi_2$, iff, $M, s \models \phi_2$ whenever, $M, s \models \phi_1$

(Refer Slide Time: 47:36)

Semantics of CTL

- 8 $M, s \models AX\phi, \text{ iff, } \forall s_1, s.t., s \rightarrow s_1; M, s_1 \models \phi$
- 9 $M, s \models EX\phi, \text{ iff, } \exists s_1, s.t., s \rightarrow s_1; M, s_1 \models \phi$
- 10 $M, s \models AG\phi, \text{ iff, } \forall \text{paths}, s_1 \rightarrow s_2 \rightarrow s_3, \dots, \forall s_i, M, s_i \models \phi$
- 11 $M, s \models EG\phi, \text{ iff, } \exists \text{path}, s_1 \rightarrow s_2 \rightarrow s_3, \dots, \forall s_i; M, s_i \models \phi$
- 12 $M, s \models AF\phi, \text{ iff, } \forall \text{paths}, s_1 \rightarrow s_2 \rightarrow s_3, \dots, \exists s_i; M, s_i \models \phi$
- 13 $M, s \models EF\phi, \text{ iff, } \exists \text{path}, s_1 \rightarrow s_2 \rightarrow s_3, \dots, \exists s_i; M, s_i \models \phi$
- 14 $M, s \models A[\phi U \psi], \text{ iff, } \forall \text{paths}, s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots, \forall s_i, M, s_i \models \phi, \forall j < i; M, s_j \models \psi$
- 15 $M, s \models E[\phi U \psi], \text{ iff, } \exists \text{path}, s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots, \forall s_i, M, s_i \models \phi, \forall j < i; M, s_j \models \psi$

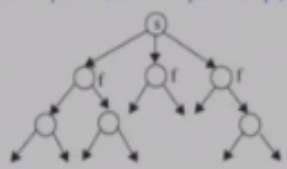
Then here, there can be F, so F or G; G is global and U. Generally, we do not define weak until or release operator for CTL, but you can define that as well. There is no such restriction. As we discussed earlier that here we can convert from one formula to another, so that means here, U2W or U2R or W2U or R2U. So, now here these formulas are defined in this way, so these are restricted formula, though restricted formulas are same.

(Refer Slide Time: 47:41)

CTL Formula

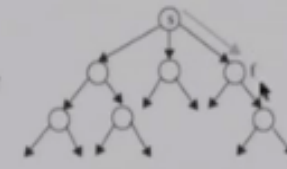
AX(f) $M, s \models AXf, \text{ iff, } \forall s_1, s.t., s \rightarrow s_1; M, s_1 \models f$

For all Paths, f holds at the next state.



EX(f)

There is a path such that f holds at the next state



$M, s \models EXf, \text{ iff, } \exists s_1, s.t., s \rightarrow s_1; M, s_1 \models f$

Whereas, the temporal formulas change a little bit. So, now here A of f, AX of f, so it says that here in all the next states, f should hold good. EX that means here it should hold good at least in one of these state.

(Refer Slide Time: 47:54)

CTL Formula

AG(f) : For all paths, f holds at every node of the path.
 $M, s \models AG\phi, \text{ iff, } \forall \text{ paths, } s_1 \rightarrow s_2 \rightarrow s_3 \dots, \forall s_i, M, s_i \models \phi$

EG(f) : There is a path along which f holds at every state.
 $M, s \models EG\phi, \text{ iff, } \exists \text{ path, } s_1 \rightarrow s_2 \rightarrow s_3 \dots, \forall s_i, M, s_i \models \phi$

AF(f) : For all paths, f holds eventually.
 $M, s \models AF\phi, \text{ iff, } \forall \text{ paths, } s_1 \rightarrow s_2 \rightarrow s_3 \dots, \exists s_i, M, s_i \models \phi$

AG should say that here in all the paths starting from states as should, I mean, f should hold good or if it is easy then here in one of the paths, this would hold good. If it is f, so that means here somewhere in the future the formula f should hold good in all the states in all the paths starting from S.

(Refer Slide Time: 48:21)

CTL Formula

EF(f)
 There is a path along which f holds eventually.

A(fUg)
 For all paths, f holds until g holds.

E(fUg)
 There is a path along which f holds until g holds.

Now if it is E, in that case here f should hold good at least in one of the paths in the future. AU is say that here; f should hold good in all the paths until g holds good. So, now here this is g hold good here; g holds good here; g holds good here. EU is defined that here f should hold good at least in one of the paths until g holds good.

(Refer Slide Time: 48:47)

Examples

1. For any state, if a request occurs, then it will eventually be acknowledged
➤ $AG(requested \rightarrow AF\ acknowledged)$
2. A process is enabled infinitely often on every computation path
➤ $AG(AF\ enabled)$
3. Whatever happens, a certain process will eventually be permanently deadlocked
➤ $AF(AG\ deadlock)$
4. From every state it is possible to get to a restart state
➤ $AG(EF\ restart)$

So, now here some of the things that we; can examples are like this, we can say if there is a request there must be acknowledgement. So, now here when there is a request here, sometimes in the future may not be in the next cycle but sometimes in the future there must be acknowledgement and this should hold good globally in arbiter or a message passing system in some protocol.

So, or here now, we can define a process is enabled infinitely often on every computations paths. So, that means here it can be enable to again and again and again, so that we can defined that here globally sometimes in the future enable signal should hold good or should become true. Now we can also, so model deadlock, deadlock says that sometimes in the future, it will instruct to or always it would be in same state.

So, that means you are deadlock, so this can model the deadlock. Then, say if you want to define that like in a processor, you can or any of the sequence cell circuit, you can always press reset and then it should come back to reset irrespective of in which state it was. So, now here that you can define that globally from every state there exists one path that can take you to the restart state.

(Refer Slide Time: 50:20)

Examples

- An upward travelling elevator at the second floor does not change its direction when it has passengers wishing to go to fifth floor

➤ $AG (\text{floor2} \wedge \text{directionup} \wedge \text{ButtonPressed5} \rightarrow A[\text{directionup} U \text{floor5}])$

- The elevator can remain idle on the third floor with its door closed

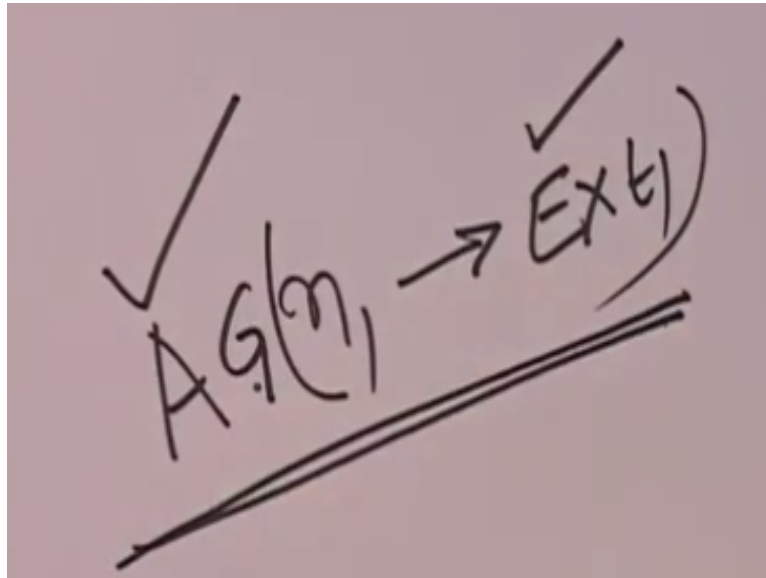
➤ $AG (\text{floor3} \wedge \text{idle} \wedge \text{doorclosed} \rightarrow EG[\text{floor3} \wedge \text{idle} \wedge \text{doorclosed}])$

So, that is the restart or reset state. Like you can also define some formula like here, if elevator is travelling upward, in that case you should keep and some; an elevator is in at floor 2 and button floor 5 is pressed, it is going upward. In that case, you should keep on going upward until floor 5th before changing their direction of the elevator or here if you want to; so now here universally globally, this would happen.

That, if it is in the floor 2 and direction is up and button floor 5 is press, in that case here, it should always keep direction up until it reaches to floor 5. The elevator can remain in idle state on the third floor with its door closed. So, now that you can say that if it is on the floor 3, and it is idle and door is close in that case here, this implies that it should continue to in this state.

So, now here one of the property that we were not able to encode in LTL now how we can encode that, so other properties you can also define in CTL, like here as I said that LTL properties are implicitly using universal quantification. So, now for the first safety property you can have universal quantification for the liveness, again here you can use the universal quantification because this should happen all the time.

(Refer Slide Time: 52:16)



Non-blocking property, we were not able to define, so that means here this was related to one path what it says is that; for every states satisfying n_1 , there should be at least one successor that can satisfy c_1 . So, that means here we were supposed to do write like this that if it is in state n_1 , there exist at least one path, where in next state is t_1 and this would hold good globally in all the states.

So, now here, why we were not able to specify this in the LTL? Because, here, we need existential quantifier and global quantifier. If you are need in only existential in that case your negation can help you but now here, you want to select A path from a set of paths. So, now here that is why you cannot specify. In the same way here, the no strict sequencing property, you can also define by converting this into their existential quantifier.

(Refer Slide Time: 53:14)

Computation Tree Logic - Equivalence

- $\neg (\varphi \wedge \psi) \equiv (\neg \varphi \vee \neg \psi)$
- $\neg (\varphi \vee \psi) \equiv (\neg \varphi \wedge \neg \psi)$
- $\neg EF \varphi \equiv AG \neg \varphi$
- $\neg AF \varphi \equiv EG \neg \varphi$
- $\neg AX \varphi \equiv EX \neg \varphi$
- $AF \varphi \equiv A[TU \varphi]$
- $EF \varphi \equiv E[TU \varphi]$

Because earlier, this was defined as negation of the property, okay. So now, if you look at all these properties will pass. Again here, there are various operators you can define the equivalence between these 2, these operators.

(Refer Slide Time: 53:23)

Computation Tree Logic - Equivalence

$$AX\phi \equiv \neg EX\neg\phi$$
$$AF\phi \equiv A[TU\phi]$$
$$EF\phi \equiv E[TU\phi]$$
$$AG\phi \equiv \neg EF\neg\phi \equiv \neg E[TU\neg\phi]$$
$$EG\phi \equiv \neg AF\neg\phi \equiv \neg A[TU\neg\phi]$$

Essential Set: AU, EU, and EX

And finally here you need a set of 3 operators in among the temporal operators and negation and conjunction or negation and disjunction as temporal operators. So, now here you need 5 operators to express any property.

(Refer Slide Time: 53:45)

Adequate Set of CTL Operators

Theorem: A set of temporal connectives in CTL is adequate if, and only if, it contains at least one of {AX, EX}, at least one of {EG, AF, AU} and EU.

And now here, so this theorem says that a set of temporal connectives in CTL is adequate if, and only, if it contains at least one of AX and or AX and EX, means from AX or EX or at least one from EG, AF, EU, so now here you need to have 3 operators AX or EX, EG or AF or AU and EU.

(Refer Slide Time: 54:17)

Checking CTL Formula

1. find all nodes at which the formula holds
2. determines whether all initial states are contained in the set of nodes

Kripke Structure

- ❖ Labeled variables are true, and the missing variables are false
- ❖ extend this labeling rule to include formulas or sub-formulas that evaluate true at the node

So, now if you want to check the CTL formula CTL, checking CTL formula is more straight forward because it is related to state not related to path, means checking for temporal formula is more difficult and what we do is; we use the labelling concept, so that means here, we start in the reverse way, we start from a point where in the formula holds good and then here we go back and try to reach to the initial state.

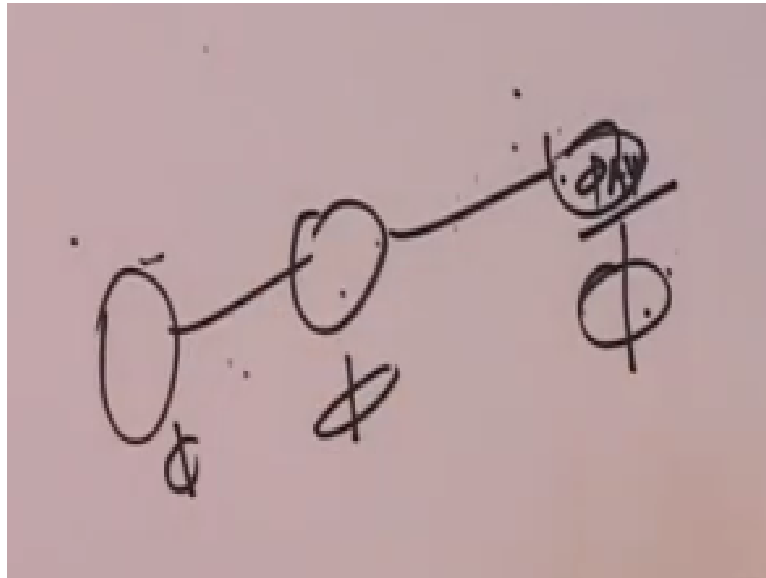
(Refer Slide Time: 54:17)

Checking CTL Formula

- consider **AND** and **NOT** operators
- if both operand formulas are true at the node, the resulting AND formula is true at the node and it is labeled at the node
- If the operand formula is not true (in other words, it is missing at a node), then the resulting NOT formula is true and it is labeled at the node
- ❖ only need to consider EXf, E(fUg), and EG(f) temporal operators

So, now we label those state, which are were in the current formula is satisfied and then we go back if the previous state of that state satisfies that formula and this way we do the backward reachability and if we happen to reach to the initial state in that case, we can say that property is all the time respected. So, now here for AND and NOT, what we need to do is here if both of the formulas are true at a node, then here resulting AND formula is true.

(Refer Slide Time: 55:32)



And so, now here say in this state, say you have these state, so here a formula ϕ holds good and that is ϕ say ϕ and ψ . Now, if this holds good in that case here, i level this formula say capital ϕ and now here because this holds good in that case here this would good in the previous state, so everywhere. But here only we need to consider, so AND and NOT are pretty straight forward.

(Refer Slide Time: 56:07)

Checking CTL Formula

- Algorithm for Checking EX(f)

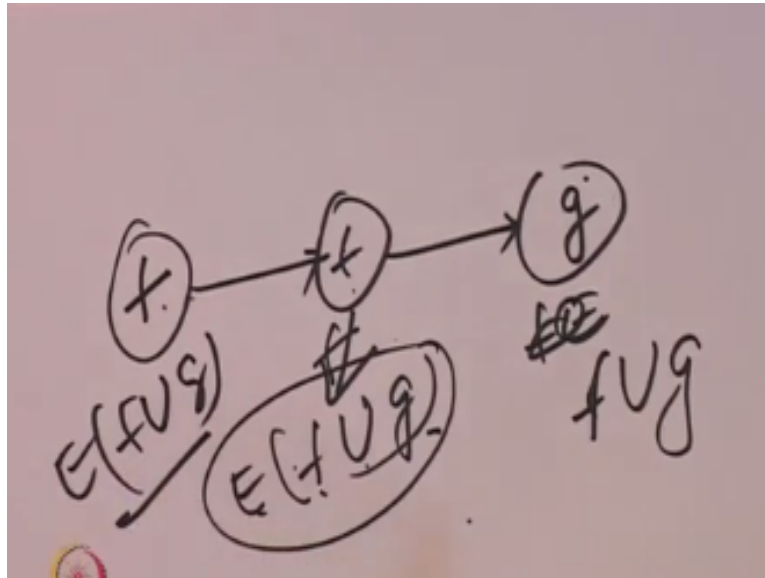
input: a Kripke structure K and a CTL formula EX(f).
output: labeling of the states where EX(f) holds

Verify_EX(f): // check CTL formula EX(f)

for each state s of K, add label EX(f) if f is labeled at a successor of s

But, here now; we need to consider EX of and EU and EG formula. So, now here EX, again here for EX, it is straight forward. Now if Xf holds good in some state in that case here, EX holds good in the previous state, so now here input is your Kripke structure, output is labelled a state transition diagram or labelled Kripke structure, where in you label EX of f. So, now here for each state, s of K, add label EX f, if f is labelled as the successor of s.

(Refer Slide Time: 56:57)



So, now here that is what I told you. EU of g; EU of g, it can be referred as now here we look at some state if these are the; these state if g is labelled here, in that case here I labelled here f; f until g in this state. Now, this is universal, so now here you look at the previous state, in this state, if the previous state is labelled as f, here f then here i can also label this as f until g and because here this is existential.

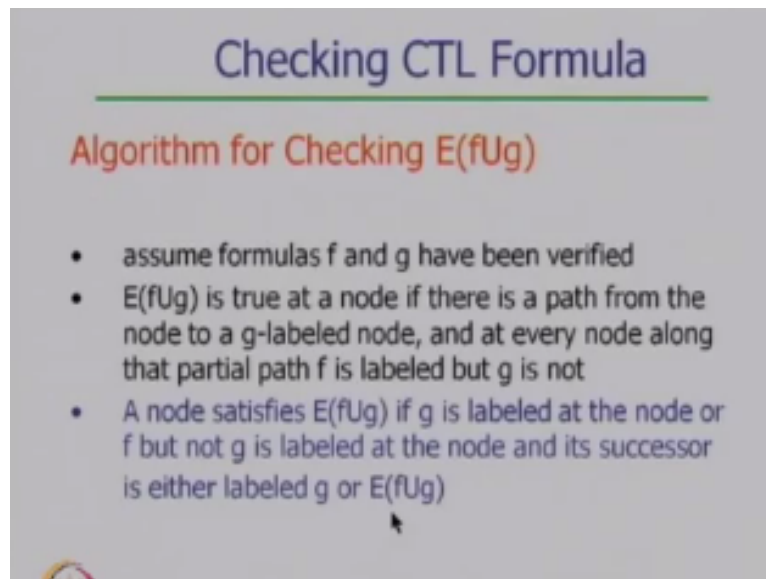
(Refer Slide Time: 57:54)

Checking CTL Formula

- **Algorithm for Checking $E(fUg)$**
input: a Kripke structure K and a CTL formula $E(fUg)$.
output: labeling of the states where $E(fUg)$ holds.
Verify_EU(f, g): // check CTL formula $E(fUg)$
 1. $M = \text{empty}$.
 2. Add label $E(fUg)$ to all states that have label g . Call this set of states L .
 3. For every state in L , if there is a predecessor, p , that is not in L , add label $E(fUg)$ to p . Add p to set M . Set M consists of newly added nodes.
 4. Set $L = M$ and $M = \text{empty}$.
 5. Repeat steps 3 and 4 until L is empty.

This means, we need to check only in one path, so in one of the successor, so now here if you look at the previous one the next state is labelled as E, f, u, g and then here earlier state also has f in that case here, you can be also labelled these as f until g, this way you keep on going. So, now here you need not to unroll your finite state machine or in other word you do not need to create complete computation tree, which is infinite long.

(Refer Slide Time: 58:29)



Checking CTL Formula

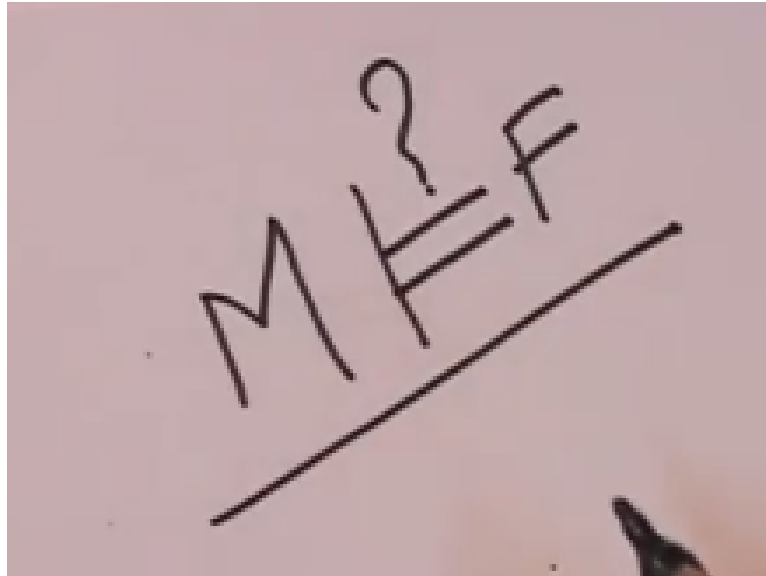
Algorithm for Checking $E(fUg)$

- assume formulas f and g have been verified
- $E(fUg)$ is true at a node if there is a path from the node to a g -labeled node, and at every node along that partial path f is labeled but g is not
- A node satisfies $E(fUg)$ if g is labeled at the node or f but not g is labeled at the node and its successor is either labeled g or $E(fUg)$

And now, here this way you keep on going back; the moment you reach to the initial state and initial state is labelled with that particular formula, you can say that formula holds good all the time. So, now here they we can dynamically reduced the complexity of verification of these formulas. Now, so this algorithm does the same thing; it says that assume the formula f and g have been verified.

So now here, $E f U g$ is true at the node. If there is a path from the node to g levelled node and every node along with the partial path f is labelled but g is not. So, now here the formula holds good, where in g holds good and now you have to check whether g holds good and f is there exit before that. So, node satisfies $E f U g$, if g is labelled at that node or f is labelled but node g is labelled at the node and its successor is either labelled as either g or f until g .

(Refer Slide Time: 59:26)



So, this way here you can verify this, so now, we are at the end. I can summarise this that you model your implementation as a transition system, you write down the properties using either CTL formula or LTL formula, say that formula is f and then here you need to check whether your formula satisfies in all the paths or in the state in which way it is referred, if it satisfies in that case here in all the cases that properties respected.

If it fails, then it will give you the counter example and counter example will help you in fixing the bug. So with this, I summarise the verification portion of these set of lectures on advanced VLSI design, good day.