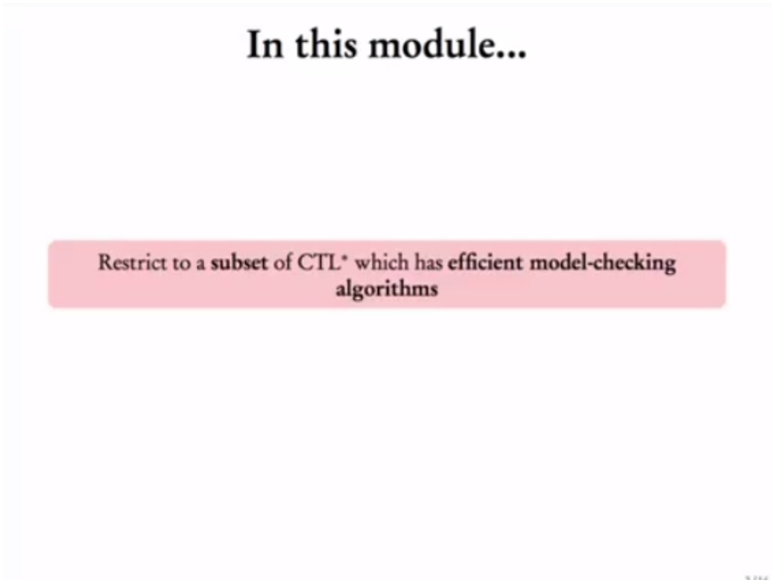
Model Checking

Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture – 43
CTL

In the last module, we have seen the logic CTL star. In this module, we will see yet another logic for talking about properties over computation trees. This logic is called CTL star.

(Refer Slide Time: 00:19)

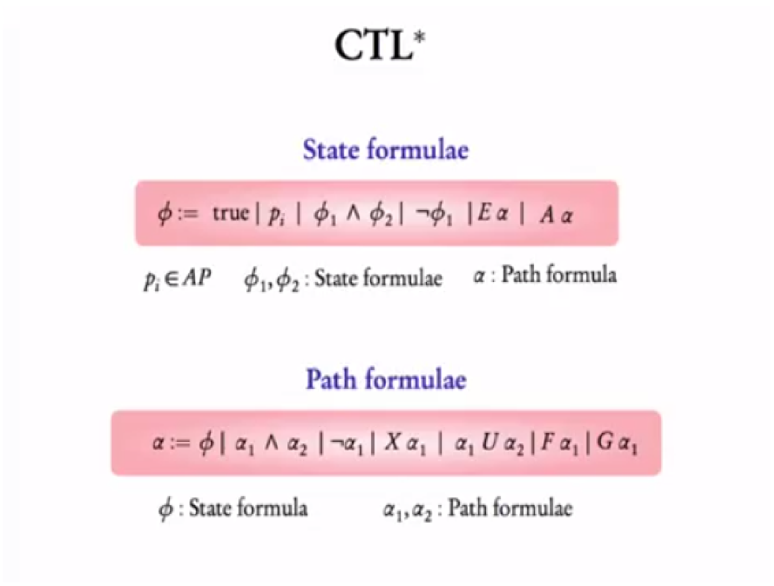


In this module...

Restrict to a **subset** of CTL* which has **efficient model-checking algorithms**

CTL will be a subset of CTL star which has efficient model-checking algorithms, so you know what CTL star is, we will eliminate certain rules in the syntax of CTL star to get some subset and these formulas will have an efficient model-checking algorithm. So the first task would be to see what this restricted subset is, we will now see what is the syntax of CTL star.

(Refer Slide Time: 00:54)



CTL*

State formulae

$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid E\,\alpha \mid A\,\alpha$

$p_i \in AP$ ϕ_1, ϕ_2 : State formulae α : Path formula

Path formulae

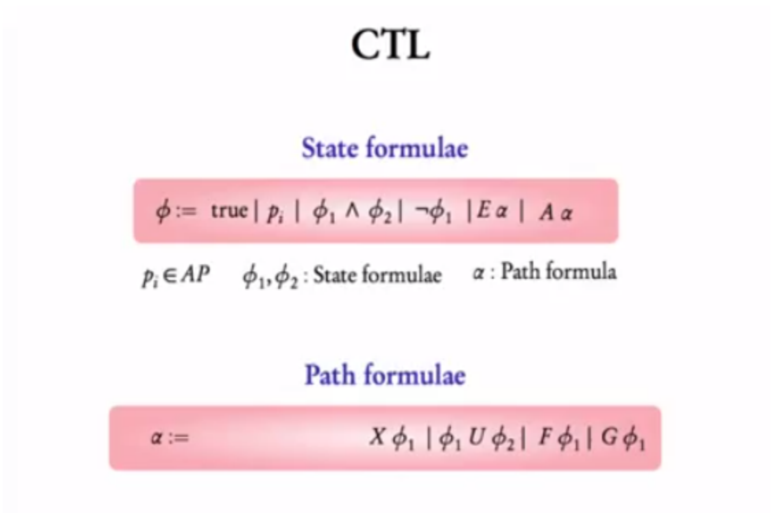
$\alpha := \phi \mid \alpha_1 \wedge \alpha_2 \mid \neg\alpha_1 \mid X\,\alpha_1 \mid \alpha_1\, U\, \alpha_2 \mid F\,\alpha_1 \mid G\,\alpha_1$

ϕ : State formula α_1, α_2 : Path formulae

So let me recall the syntax of CTL star first. We have state formulae and path formulae. State formulae are formed by this set of rules true Pi phi 1 and phi 2 where both phi 1 and phi 2 are state formulae not of phi 1 and these are the additional operators for talking about trees. This says that there exists a path that satisfies a path formula. All part satisfy this path formula and path formulae are formed using these set of rules.

A state formula can be considered as a path formula and the semantics is that a path satisfies phi if the initial state satisfies phi. If alpha 1 and alpha 2 are path formulae the and of them is a path formula and the knot of a path formula is a path formula and you have the temporal operatives x, u, f, g okay. So x of a path formula alpha 1 until alpha 2 where both alpha 1 and alpha 2 are path formulae, f of alpha 1, g alpha 1 okay.

(Refer Slide Time: 02:32)



CTL

State formulae

$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid E\,\alpha \mid A\,\alpha$

$p_i \in AP$ ϕ_1, ϕ_2 : State formulae α : Path formula

Path formulae

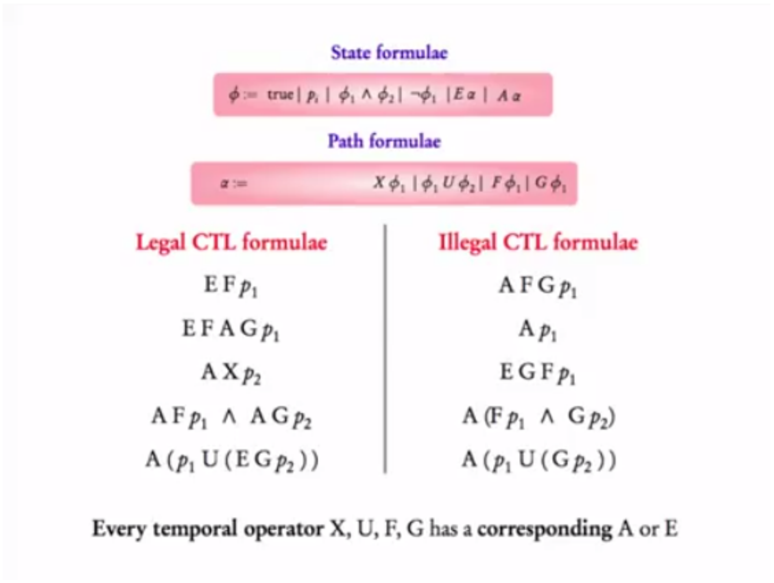
$\alpha := X\,\phi_1 \mid \phi_1\, U\, \phi_2 \mid F\,\phi_1 \mid G\,\phi_1$

So now let us see what are the rules that we are going to eliminate to get CTL star. Firstly, the rules which said that alpha could be a state formula or you can form and off path formulae and knot of path formulae that has been removed. So these three letters first removed. We will see what are the formulae that go away because of this but before that there are more things to be changed, this says that x of alpha 1.

When we had this, we could form formulae like x of x of p1 for instance because you could have had x of alpha 1 where alpha 1 could have been x of something and that something could be a state formula which can be p1. So now since we have removed this, we need to modify this as well to get a base. We will change alpha 1 to phi 1. So instead of saying x of a path formula you restrict x of a state formula. So you can say x of p1 okay.

You cannot say x of x of p1 for instance. Let me do the rest and explain with examples what are the kinds of formulae that are possible. So yet again instead of writing alpha 1 until alpha 2, we can say a state formula is true until some other state formula is true. About F alpha 1that changes to F phi 1. Similarly, G alpha 1 changes to G phi 1. So this is going to be CTL. Let us now see what are the kinds of formulae that are possible in CTL.

(Refer Slide Time: 04:34)



State formulae

$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid E\,\alpha \mid A\,\alpha$

Path formulae

$\alpha := X\,\phi_1 \mid \phi_1\, U\, \phi_2 \mid F\,\phi_1 \mid G\,\phi_1$

Legal CTL formulae	Illegal CTL formulae
$\mathrm{E\,F}\,p_1$	$\mathrm{A\,F\,G}\,p_1$
$\mathrm{E\,F\,A\,G}\,p_1$	$\mathrm{A}\,p_1$
$\mathrm{A\,X}\,p_2$	$\mathrm{E\,G\,F}\,p_1$
$\mathrm{A\,F}\,p_1 \wedge \mathrm{A\,G}\,p_2$	$\mathrm{A}\,(\mathrm{F}\,p_1 \wedge \mathrm{G}\,p_2)$
$\mathrm{A}\,(p_1\,\mathrm{U}\,(\mathrm{E\,G}\,p_2))$	$\mathrm{A}\,(p_1\,\mathrm{U}\,(\mathrm{G}\,p_2))$

Every temporal operator X, U, F, G has a corresponding A or E

So this is the syntax of CTL. Let us look at legal CTL formulae. You can form E F p1. For example, phi could be E alpha where alpha could be one of these F of phi 1 where phi 1 is p1. So E F p1 is a valid formula that can be formed out of these rules. E F A G p1, so see phi could be E of a path formula, here the path formula is F of A G p1 where A G p1 is a state formula.

See you can form A G p1, A of alpha and alpha could be G phi 1 where phi 1 is p1. So A G p1 is a state formula and then you form one more state formula like this, E of alpha where alpha is F of some state formula which is A G p1. So you see E F A G p1 is a valid, is a legal CTL formula. A X p2, this is simple. So A of alpha where alpha is X of phi 1 where phi 1 can be p2 okay.

You can form and of two legal CTL formulae. So you can say A F p1, which is possible A F p1 A G p2 and you can form the and of them. Let us now see some formulae that cannot be formed using these rules, A F G p1. Remember that A F G p1 is a legal CTL star formula, this says that all parts satisfy the path property F G p1.

So this was a valid CTL star formula but this is not a legal CTL formula because you see phi can be A of alpha okay. So if this is formed from the set of rules, then it has to use the rule that phi can be A alpha. Now alpha could be F of phi 1 okay. Now we need to get G p1 here but this is not possible. No state formula starts with G right. You see any state formula will start with either E or A or just the atomic propositions and then you can have knots and ions of that.

You cannot start the state formula with G. So this is not a legal CTL formula although it is a CTL star formula, this is not a legal CTL formula. A p1, for example look at this, if A p1 is formed from this grammar, then it should have used the rule phi can be A alpha where alpha is the path formula. Now look at alpha, alpha needs to have the temporal operators X U F or G but this is not the case.

In the case of CTL star, we had a phi here, which could derive A p1 but that has been eliminated. So A p1 is not a legal CTL formula. E G F p1 for the same reasons as this E G F p1 cannot be derived using these set of rules. So E G F p1 could be derived from here, you should have used the rule E alpha but then alpha is a path formula and since you have G F p1, it should have started with G phi 1 but then phi 1 cannot start with F okay.

What about this, A of F p1 and G p2 can we do that. If this was a legal CTL formula then the rule phi goes to A alpha has to be used where alpha is a path formula. But then we do not have the and operator here at all. So this have been a path formula then you should have had the rule alpha 1 and alpha 2. So this cannot be a legal CTL formula.

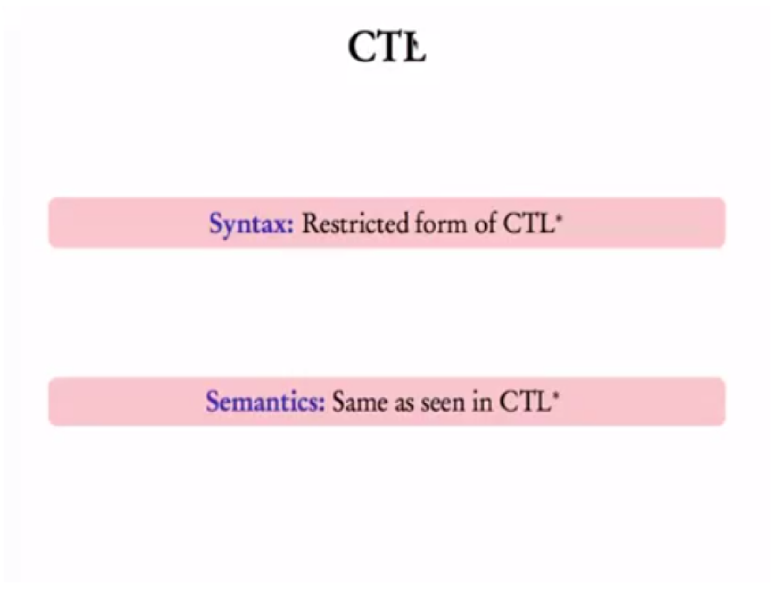
Let us look at this A of p1 until E G p2, is this a legal CTL formula, let us see. The rule phi goes to A alpha has to be used where alpha is this formula which is of the form phi 1 until phi 2 where phi 1 is p1 phi 2 is E G p2. Let us see if E G p2 can be derived as a state formula, yes. You use the rule E alpha and then take alpha to be G phi 1 and phi 1 is p2. So you have A of p1 until E G p2 to be a legal CTL formula.

If you see this A is associated with this U, when you do A of alpha, alpha is phi 1 until phi 2, so this until is associated with this A. Similarly, this was a state formula E alpha where E is associated with G okay. However, this cannot be a valid legal CTL formula because this is p1 until G p2. So A alpha phi 1 is p1 can G p2 be derived as a state formula, no, since the state formula cannot start with G, this is not a legal CTL formula.

This is the thing, every temporal operator has a corresponding A or E. So if you see this, this is what I was explaining before A is spared with this until E is spared with this G. Here A is spared with F, A is spared with G. Similarly, here this A is spared with X. This is how the rules will let you form formulae because the state formula would if it has an E or an A, it should have one of these temporal operators associated with this because the formula is of the form E alpha A alpha where alpha needs to have one of these temporal operators.

Here you see, there are two temporal operators but only one path quantification operator A, so this cannot be a legal CTL formula. Same is the case of this one as well. You have two temporal operators but only one path quantification operator. So a legal CTL formula should have corresponding temporal operators for every A and E okay. Please try to go through certain examples and check for yourself if that can be a legal CTL formula or not.

(Refer Slide Time: 12:30)



CTL

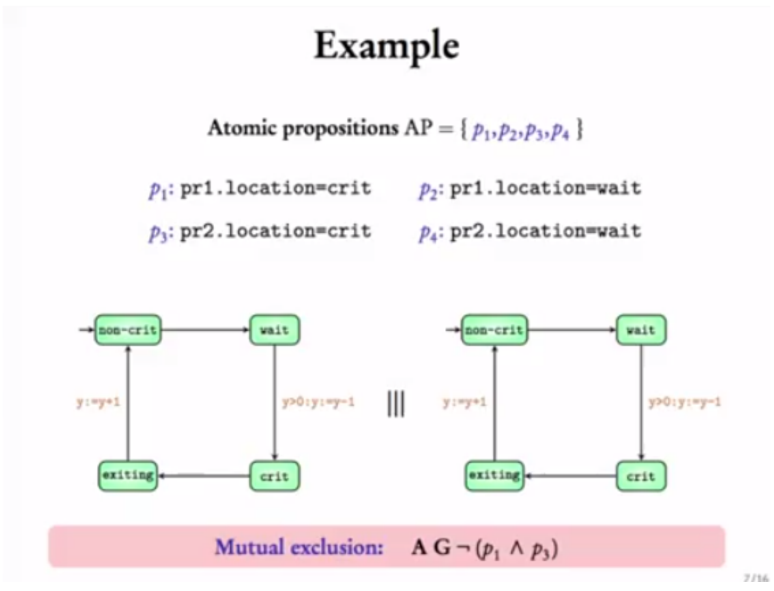
Syntax: Restricted form of CTL*

Semantics: Same as seen in CTL*

So here is CTL. In the last slide we saw the syntax, it is a restricted form of CTL star and why we need to do this restriction is because this restriction will give better algorithms and if are properties can be expressed using CTL itself then the fact that it has efficient algorithms can be exploited. The semantics that is the meaning of CTL formula, it was the same as in the case of CTL star.

As you see every CTL formula is also a CTL star formula. So you look at the semantics of that formula that we discussed in the previous module. So this means that there exists a path where at some point of time p1 is true. Similarly, there exists a path where at some point of time you reach a state whose sub tree satisfies p1 in every role okay. You can work out the semantics for other formulae similarly.

(Refer Slide Time: 13:35)



Example

Atomic propositions AP = { p_1, p_2, p_3, p_4 }

p_1: `pr1.location=crit` p_2: `pr1.location=wait`

p_3: `pr2.location=crit` p_4: `pr2.location=wait`

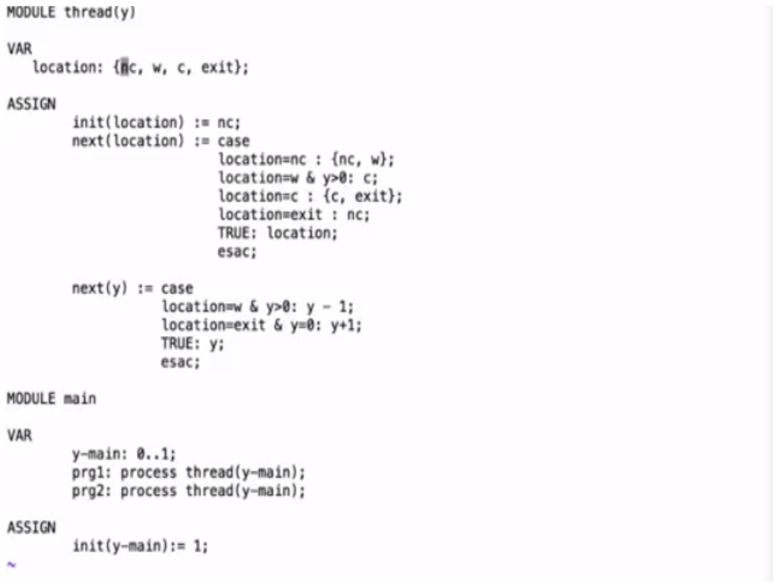
Mutual exclusion: $A\,G\,\neg(p_1 \wedge p_3)$

7/16

So let us look at the mutex example, you have two programmes, they are identical threads and they are accessing a shared resource and this y makes sure that the critical section is not handled by both the processes at the same time. We have seen this example multiple times. Now the mutual exclusion property is that at no point of time both the processes are in the critical location simultaneously. So how can we write this using CTL.

You look at the computation tree, in all parts, in all states this has to be true. A G it is not the case that both process 1 and process 2 are in there critical locations. So p1 is the atomic proposition which says that process 1 is in its critical location, p3 is the atomic proposition that says that process 2 is in its critical location and A G knot of p1 and p3 is the property expressing the fact that both the processors are not in there critical sections simultaneously. Let us now try to check the CTL property in NuSMV.

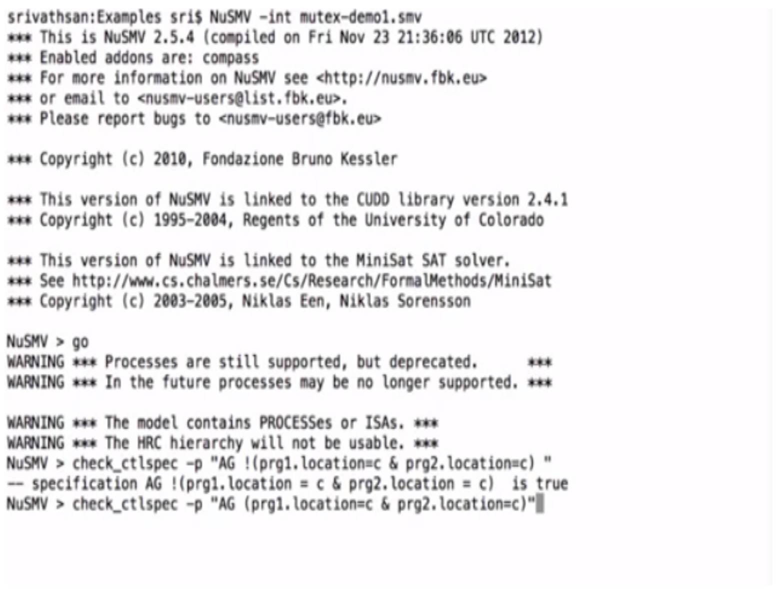
(Refer Slide Time: 15:02)



```
MODULE thread(y)

VAR
   location: {nc, w, c, exit};

ASSIGN
       init(location) := nc;
       next(location) := case
                      location=nc : {nc, w};
                      location=w & y>0: c;
                      location=c : {c, exit};
                      location=exit : nc;
                      TRUE: location;
                      esac;

       next(y) := case
                 location=w & y>0: y - 1;
                 location=exit & y=0: y+1;
                 TRUE: y;
                 esac;

MODULE main

VAR
       y-main: 0..1;
       prg1: process thread(y-main);
       prg2: process thread(y-main);

ASSIGN
       init(y-main):= 1;
~
```

So this is the mutex programme and this module thread is defining the programme graph and you have the main module where you have the variable y and two processes prg1 and prg2 okay and the initial value of y is assigned to be 1. The locations in this programme graph are non critical waiting critical and exit. We had checked LTL specifications. Let us now try to check it using CTL.

(Refer Slide Time: 15:40)



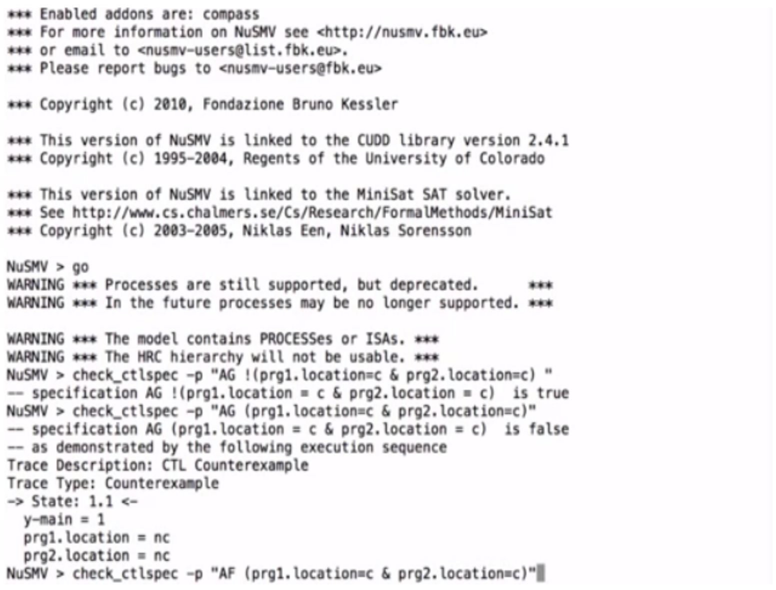
```
srivathsan:Examples sri$ NuSMV -int mutex-demo1.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > check_ctlspec -p "AG !(prg1.location=c & prg2.location=c) "
-- specification AG !(prg1.location = c & prg2.location = c)  is true
NuSMV > check_ctlspec -p "AG (prg1.location=c & prg2.location=c)"
```

As usual we do NuSMV minus int, the name of the file and say go. Remember in LTL we used the command check LTL spec, here we will do check underscore CTL spec minus p AG, there should be no space between A and G, it is not the case that prg1 dot location equal to c and prg2 dot location equal to c.

Let us now check if this property is true. It says that the property is true. Recall that whenever we use to check LTL formulae, if the specification is false then NuSMV returned a counter example there it was possible because a counter example would be an infinite trace. Now suppose you check a property on a tree, what would counter examples be.

So in general you cannot expect counter examples for all CTL properties, however, for certain CTL properties where the counter example can be given as a trace, NuSMV will do it. For example, if we check the CTL specification without the knot here. So prg1 dot location equal to c and prg2 dot location equal to c. Clearly this should be false. Let us see what NuSMV tells.

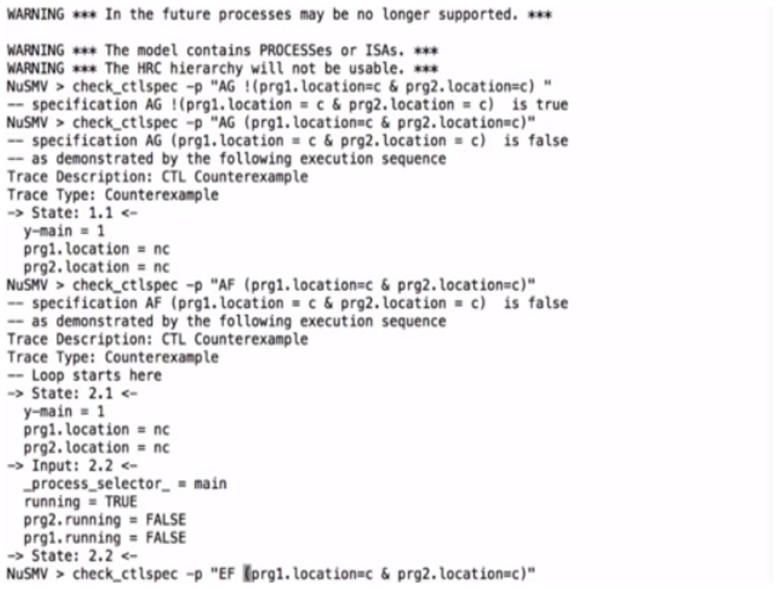
(Refer Slide Time: 17:38)



```
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > check_ctlspec -p "AG !(prg1.location=c & prg2.location=c) "
-- specification AG !(prg1.location = c & prg2.location = c)  is true
NuSMV > check_ctlspec -p "AG (prg1.location=c & prg2.location=c)"
-- specification AG (prg1.location = c & prg2.location = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
NuSMV > check_ctlspec -p "AF (prg1.location=c & prg2.location=c)"
```

Yes, it says that the specification is false and the counter example is given to be the initial state itself. The initial state itself does not satisfy this condition. We asked AG prg1 dot location equal to c and prg2 dot location equal to c and a counter example to this property is that the initial state already does not satisfy this condition. Suppose we check AF, is it possible that rather in all parts is it possible to reach a location like this. Let us see what NuSMV says.

(Refer Slide Time: 18:30)

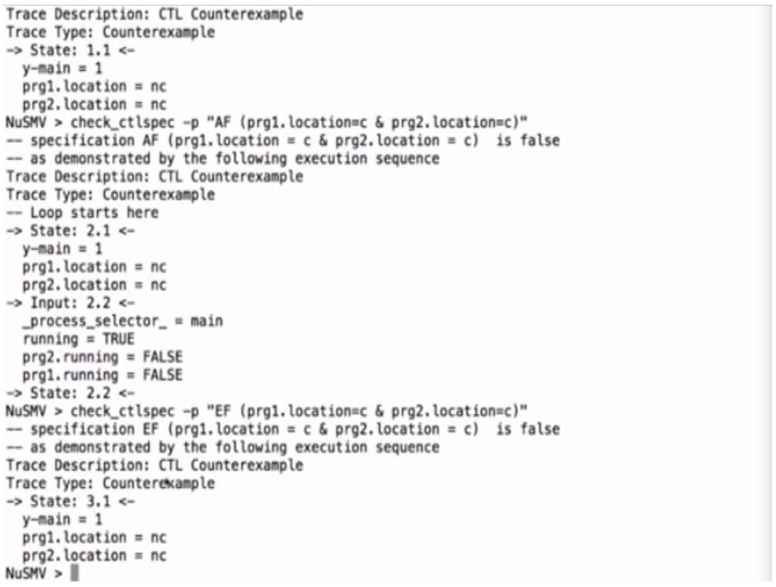


```
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > check_ctlspec -p "AG !(prg1.location=c & prg2.location=c) "
-- specification AG !(prg1.location = c & prg2.location = c)  is true
NuSMV > check_ctlspec -p "AG (prg1.location=c & prg2.location=c)"
-- specification AG (prg1.location = c & prg2.location = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
NuSMV > check_ctlspec -p "AF (prg1.location=c & prg2.location=c)"
-- specification AF (prg1.location = c & prg2.location = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
-> Input: 2.2 <-
  _process_selector_ = main
  running = TRUE
  prg2.running = FALSE
  prg1.running = FALSE
-> State: 2.2 <-
NuSMV > check_ctlspec -p "EF (prg1.location=c & prg2.location=c)"
```

Of course, the specification is false, now the counter example to this is going to be a trace because this condition says is it true that in all parts at some point of time you reach this condition, if it is not true, that means NuSMV should exhibit a path where this condition is never true right and that is what it does. It shows a path here. So the loop starts here. You have nc nc, both of them are in critical locations and there is a self loop here.

So in this path, there is an infinite path which keeps sticking to this location itself and no where along this path is this condition true. So here NuSMV gives an infinite trace as a counter example. However, suppose we ask EF, does there exist a path where this is true. Let us see what NuSMV says.

(Refer Slide Time: 19:41)

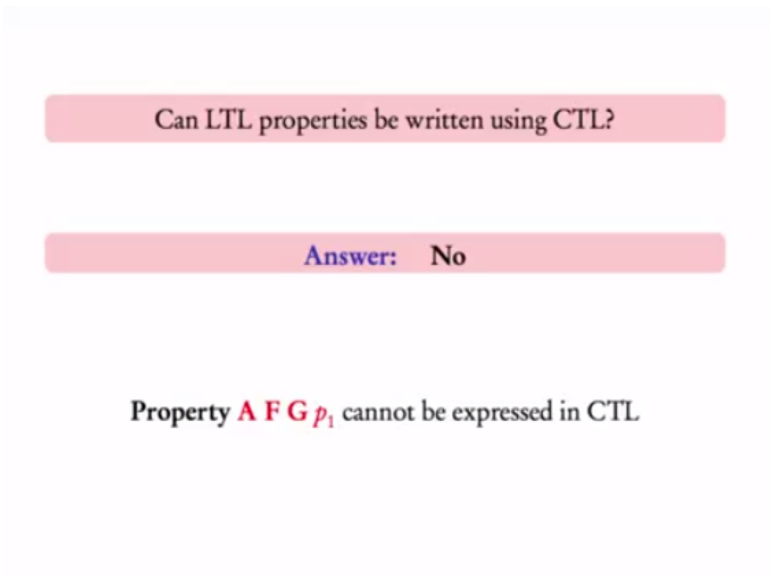


```
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
NuSMV > check_ctlspec -p "AF (prg1.location=c & prg2.location=c)"
-- specification AF (prg1.location = c & prg2.location = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
-> Input: 2.2 <-
  _process_selector_ = main
  running = TRUE
  prg2.running = FALSE
  prg1.running = FALSE
-> State: 2.2 <-
NuSMV > check_ctlspec -p "EF (prg1.location=c & prg2.location=c)"
-- specification EF (prg1.location = c & prg2.location = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 3.1 <-
  y-main = 1
  prg1.location = nc
  prg2.location = nc
NuSMV > 
```

It says that no, that is not true, that means what in all parts F of this is not true. So here the counter example does not make sense because here the counter example is the entire tree itself. So we cannot learn much out of the counter example here. So AF, AG there is some hope for learning something from the counter example.

For EF, the counter example is just the entire tree. It just says that no where this condition is true okay. That was a digression to show how one can check CTL properties on NuSMV specifications. Note that this was a property which we could already express in LTL itself.

(Refer Slide Time: 20:46)



Can LTL properties be written using CTL?

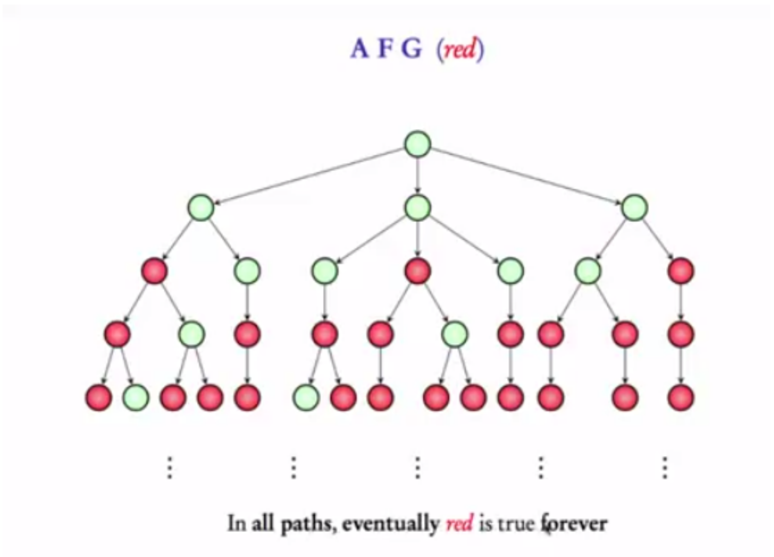
Answer: No

Property **A F G** p_1 cannot be expressed in CTL

Can all LTL properties be written using CTL, and the answer is no and this is one such property, this says that, this is a CTL star property. This says that in all parts F G p1 is true. So this can be expressed using LTL formula F G p1. A transition system satisfies F G P1 if all parts satisfy F G p1 and what is F G p1, eventually p1 is true forever. So you reach a point from which p1 is always true.

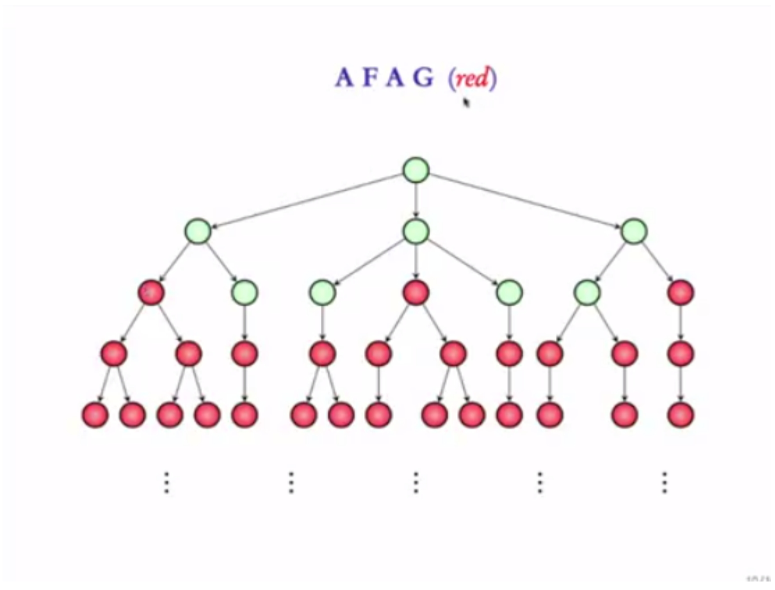
Note that this is first of all not a CTL formula because there are two temporal operators but only one path quantification operator. However, the proof that this property cannot be expressed by any formula of CTL is not easy. I will not give the full proof but I will give some intuition as to why this cannot be captured by certain CTL formulae.

(Refer Slide Time: 22:02)



So first of all, A F G p1, let p1 be red. A F G red means that in all parts you reach a point from which red is always true. So along this path in future sometime you would reach a point from where red will always be true. Along this path, perhaps this is the position from which red is always true and so on. So you look at any path, you will reach a point from which red is always true okay. This is the property A F G red. In all parts, eventually red is true forever.

(Refer Slide Time: 22:46)

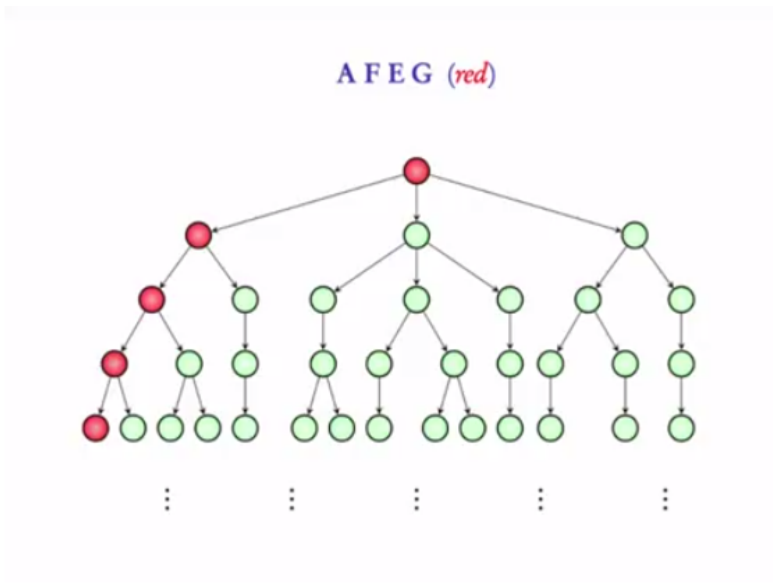


What about A F A G, this a CTL formula, does it represent A F G, let us see. A F A G red, this says that in all parts you reach a point from which the entire sub tree should satisfy red. This is a stronger condition. Look at A F G red, you can have a tree where these nodes are never coloured red. This one, this one, this one, this one, and yet it satisfies A F G red.

For example, for this path may be the eventual conditions starts from here. For this path, the eventual conditions start from the sub tree. For this path, the eventual condition start later in the sub tree. So if you look at these nodes I mean you look at there are infinitely many nodes, which are children of the left most path.

If you look at this entire set of nodes, it is possible that the set of nodes is not coloured red but still the tree satisfies A F G red. This cannot happen here because at some point you will reach a node such that the entire sub tree below it satisfies red. So in particular these nodes will have to be coloured red okay. So this is a stronger condition.

(Refer Slide Time: 24:27)

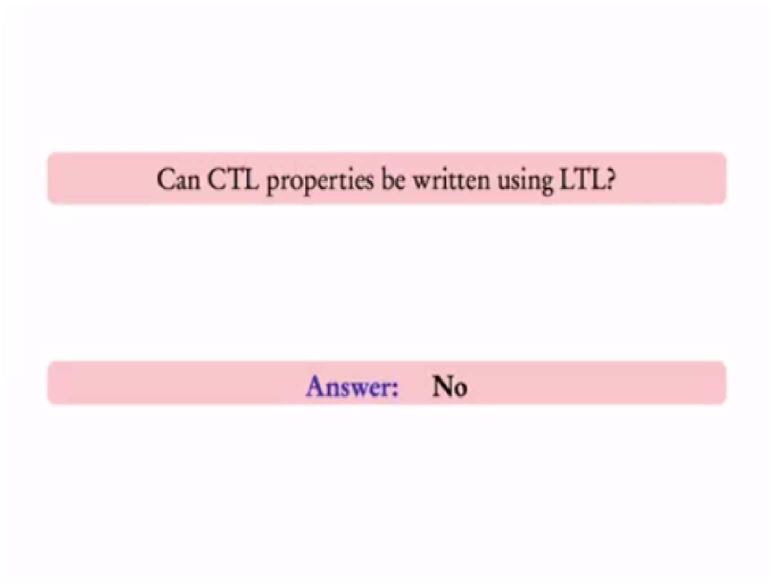


Let us now look at another formula A F E G. Is A F E G equivalent to A F G, it will not be the case. Note that A F E G is a legal CTL formula, however, look at this tree, this left most path is coloured red fully. Now look at any path, it has a state where E G red is true. So look at any path in this tree since the initial state has a path starting from with which is fully coloured red in any part the initial state satisfies E G red.

So this tree satisfies A F E G red. Look at this path, it satisfies F E G red because the initial state satisfies E G red similarly for all parts. However, clearly this does not satisfy A F G red because A F G red says that in all parts eventually red is true forever. So A F G red is a formula that can be expressed using CTL star and also LTL because it says that all parts should eventually see p1 forever.

Here p1 was red and this is equivalent to the LTL formula F G p1. However, this cannot be written using CTL. The proof is complicated, here I have given you two examples of CTL formulae which are not equivalent to A F G red. So this is the summary, A F G p1 cannot be expressed in CTL.

(Refer Slide Time: 26:32)

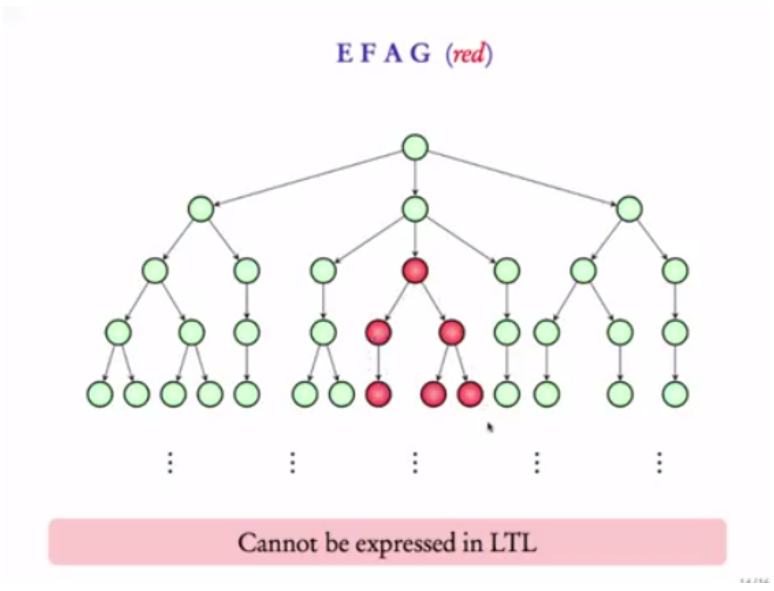


Can CTL properties be written using LTL?

Answer: No

What about the reverse question, can all CTL properties be written using LTL. Here we asked a question can all LTL properties be written using CTL in which case we would be using only CTL unfortunately the answer is no. Now we asked the reverse question, what if every CTL property can be written using LTL then we do not have to use CTL at all and unfortunately, the answer is no here as well.

(Refer Slide Time: 27:04)

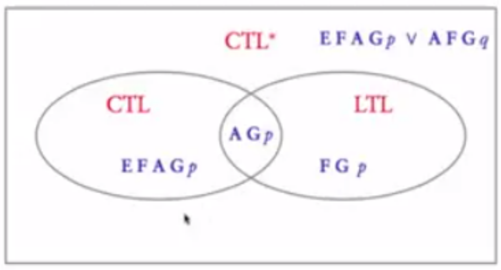


E F A G (*red*)

Cannot be expressed in LTL

And that is because of this CTL formula, we also saw this in CTL star. So E F A G red, this is a CTL formula which says that there exist a path which reaches a state from which the entire sub tree satisfies red and such a property cannot be expressed using LTL.

(Refer Slide Time: 27:25)



So here is the relative expressive power of the three logics that we know. LTL is linear temporal logic. CTL is computation tree logic and CTL star is a super set of both of them. So there are some properties which can be expressed using both LTL and CTL. For example, A G p, I mean, in all paths, in all states p is true. This is the in variant property which can be expressed using both LTL and CTL. E F A G p.

The one that we saw in the previous slide which says that there exists a path from which the entire sub tree below it satisfies p. This can be expressed using CTL but not in LTL and we also saw that F G p, this can be expressed using LTL but this condition cannot be checked using CTL, however, CTL star contains both LTL and CTL. Now, is there a formula in CTL star which cannot be expressed by both CTL and LTL.

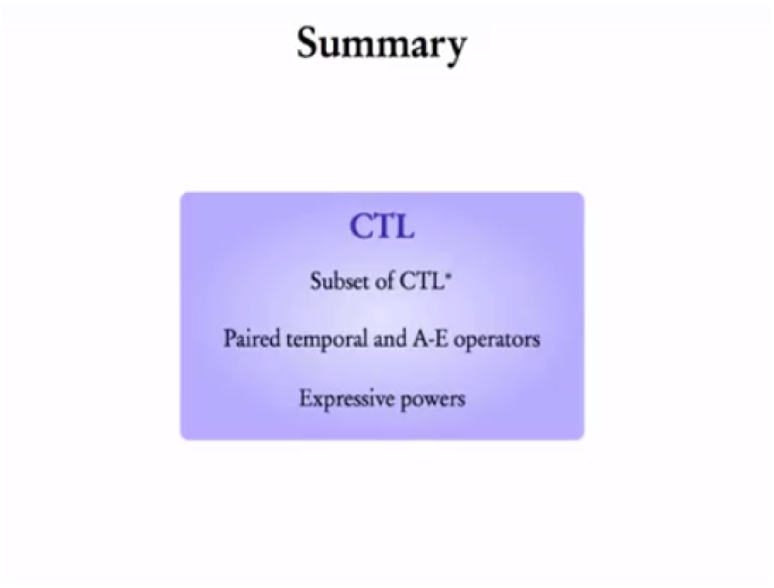
Yes, take the OR of this. E F A G p or A F G of some other proposition q. Now this cannot be expressed using CTL why because suppose this where expressible using CTL. So this gives us the set of trees that satisfy either this or this. Now suppose there was a CTL formula for this, we know that this is a CTL formula and we can take a knot of this. So look at the CTL formula for this and knot of this.

That will be another CTL formula and that will be the formula for A F G q because you take this, I mean you take the CTL formula which expresses this if at all there is one and knot of this would mean that this has to be true and so this is giving us the CTL formula for A F G q and that is not possible. Similarly, if there had been an LTL formula for this you can do a similar trick to say that an LTL formula for this would give us an LTL formula for this.

But however, we know that that is not possible okay. So we have seen a lot of logics in this unit. So CTL is essentially for talking about computation trees of transition systems and CTL is useful because the algorithms for CTL are more efficient. However, writing CTL specifications is slightly more difficult than LTL. So if you are comfortable writing LTL specifications.

And if they are indeed enough for your purpose then you might as well use LTL however, if there are properties like this which are needed then CTL should be used.

(Refer Slide Time: 30:49)



Summary

CTL

Subset of CTL*

Paired temporal and A-E operators

Expressive powers

Here is a summary, CTL is the subset of CTL star where the temporal operators and the A-E operators come in pairs. So we have seen the syntax of that and we gave a comparison of the expressive powers of CTL, CTL star and LTL. So CTL star is a logic which subsumes everything and algorithms for CTL star are difficult. Similarly, algorithms for LTL are exponential.

However, we will see in the next unit that algorithms for CTL are more efficient than both LTL and CTL star. So this is a fragment of CTL star with efficient model-checking algorithms. Writing CTL specifications is slightly difficult than LTL. So that is why we have tried to give a lot of examples in this unit. You can try to go through them and become more comfortable in writing CTL specifications.