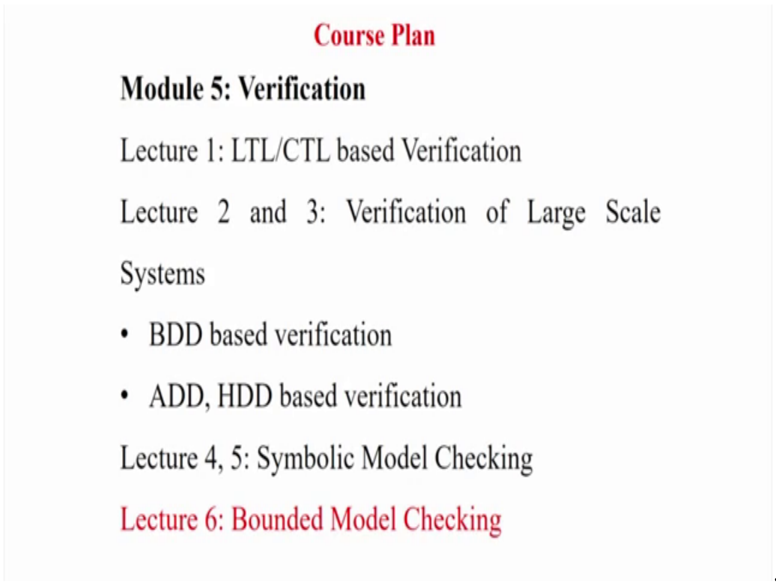
Optimization Techniques for Digital VLSI Design

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

Lecture - 24
Verification: Bounded Model Checking

So, welcome to the last lecture on the module on verification. So, basically till now if you look at our course, we have in the in this module on verification.

(Refer Slide Time: 00:33)



Course Plan

Module 5: Verification

Lecture 1: LTL/CTL based Verification

Lecture 2 and 3: Verification of Large Scale Systems

- BDD based verification
- ADD, HDD based verification

Lecture 4, 5: Symbolic Model Checking

Lecture 6: Bounded Model Checking

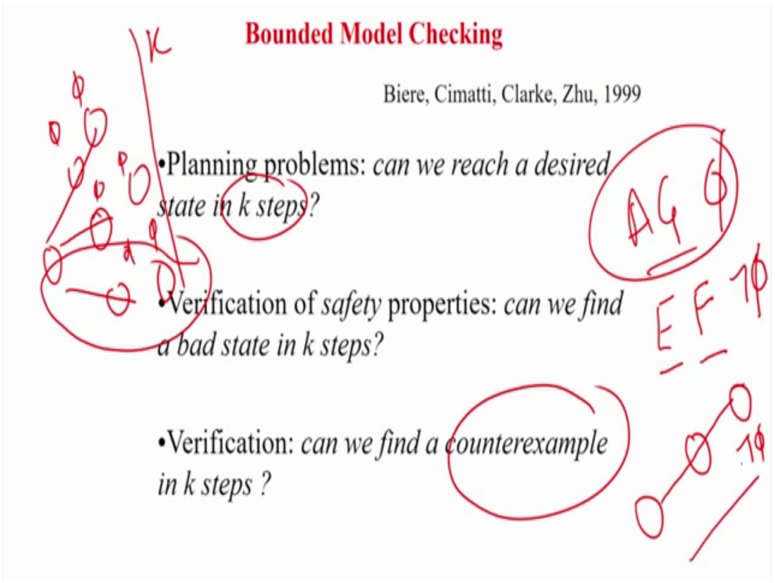
We were mainly discussing that given a system, how we can formally verify that the model actually behaves our desired intent? So, we started with the basic LTL and CTL based model verification, then there we have seen basically the main problem, which actually brings down the scalability issue is modeling the system in terms of state space. So, given a system the way to model it is the major bottleneck.

So, then we have seen how we can go for verifying large scale systems. So, one way of improving scalability is that you can use a binary decision diagram and use symbolic model checking. So, basically the idea is that instead of modeling explicit states based enumeration we try to model using BDD. So, where lot of redundant variables are eliminated and then all our labeling algorithms, which basically is the heart of CTL and LTL model checking we can easily do on the BDD itself. So, that is actually called symbolic model checking which actually optimizes these verification paradigm from normal state space modeling to a symbolic of 2 BDD based model checking.

So, where we can go from a jump of small scale system to somewhat large scale system, but then we have seen that if you really want to go for noc or soc kind of structures bit level modeling will not at all help you. So, there we had a entire lecture on a high level decision diagrams, like arithmetic decision diagrams, high level decision diagrams, etcetera, but the problem there was that we do not have very good labeling and modeling model checking algorithms on those high level decision diagrams.

So, that is actually a way of is a part of research paradigm well we are to where people are trying to go for symbolic kind of model checking not on BDDS which are at binary bit level or binary level written on the high level models, but still as I was discussing on that lecture also that is not very well matured today basically we are going to see another way of optimization that is actually called bounded model checking. So, what is the core idea?

(Refer Slide Time: 02:26)



Bounded Model Checking

Biere, Cimatti, Clarke, Zhu, 1999

- Planning problems: *can we reach a desired state in k steps?*
- Verification of *safety* properties: *can we find a bad state in k steps?*
- Verification: *can we find a counterexample in k steps ?*

So, in case of symbolic model checking what we are doing we were at the gate level or we were at the bit level in bounded model checking also we are at the lower level of bit level only, because we are when we are going for model checking there will be a labeling algorithms. So, which we are not very well matured techniques for the high level diagram. So, when we talk of bounded model checking also we are also at the lower level of enumeration, but in case of a symbolic model checking what we have seen we have eliminated or handle complexity by using binary decision diagram.

So, that redundant variables are eliminated. So, we have a lot of compression in case of model bounded model checking we think of in a drastic other direction. So, in that case what we do see for example, we have to find out a safety property, that in all paths globally something it should hold like in all paths globally some safety property some safety property can be like, you have a microwave the heating is on correct and the door should not be door should be closed that is safety property which should be verified in all these states.

Now, if you are going (Refer Time: 03:26) all paths globally. So, if you are thinking of all paths globally; that means, you have to check that propositional formula in all the all the states, but rather in bounded model checking what we will do rather than checking maybe for all long long traces from the initial stage state, we are going to bound our search to a restricted set. That is maybe if I say that bound is 2 in this case planning problems we can reach the desired state in k steps? So, you put a bound that from the initial state we reach and try to verify the properties as given in the CTL or LTL formula, but we bound the search space or search length face of length k we are already going to search.

So, what we are going to do basically we are having an optimization by not searching the entire long very very long traces where we are searching in a bound, what are the advantage and disadvantage obvious disadvantage is you are losing all the completeness, because there may be a bug or there may be a problem in some say if it is very long further down the line in the trace as you not be able to find the because we are putting a bound. But advantage is that we are we will be handle larger systems, because now we are not searching for a very large state space. So, you not even model those long we are not even going to model the state space or transition relations at the further down the lines.

So, there is some tradeoff in case of BDD or symbolic model checking we are only having a advantage that we are improving the complexity, but we are not losing anything because if you are modeling any system by bdds we are not losing any information, rather we are only eliminating the redundant variables. So, there is no loss in any kind of completeness or correctness, but here we are going to lose on that ground basically, but anyway the idea is that for may for most of the general cases you find the bugs very early in a very very rare case that the bug will be some something 10 000 states down the line, rather it is very provable that the bugs will be found very in the states which are having shorter traces trace length from the initial state.

So, on that core idea bounded model checking is standing and 1 more thing before we start basically bounded model checking generally understands in once in a counter example idea. So, what is the counter example idea I will give you of basic gist like we are saying for all paths globally some formula phi should hold that is a safety property as the example, I have given you microwave is running there is heating is on, but at the same time the door is closed. So, the safety property should hold everywhere all paths G.

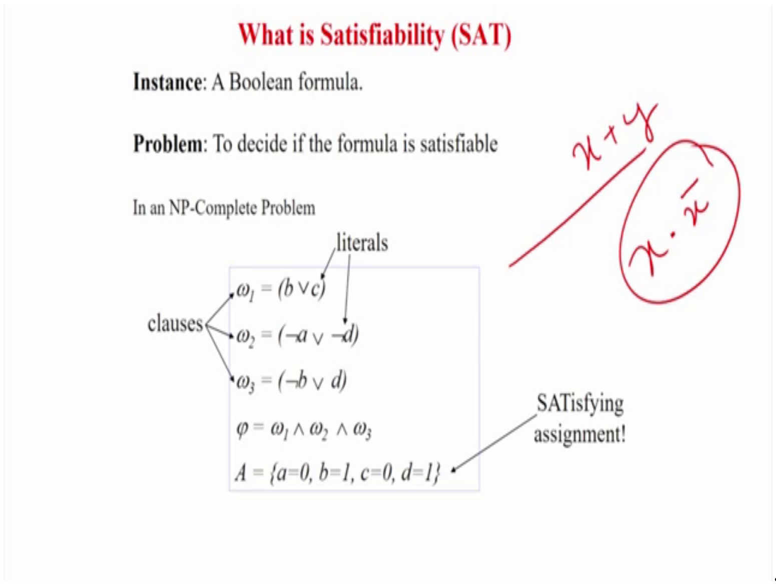
So, now in this case if you are even if you are going for bounded model checking for all paths of length k from the initial states you have to verify, but I can do a slightly different way just thing about it there exists a path where in future not of phi; that means, what; that means, you are saying the first safety property says that all paths if you take maybe I have a bound, which have talking a bounded model checking.

So, everywhere this phi should hold this is I am going to verify. So, of course, whatever formula you develop you have to search the entire space at least up to bound k, but if I just think of a counter example if I can somehow find at least 1 path maybe same this 1 where in future, but here also as I told you bounded model checking in future cannot be infinite these are bound on the future that is what is a bounded model checking if I say that in all paths in future.

So, basically in a normal model checking sense the future can be infinite, but in bounded model checking we restrict the future length. So, what even we try to see if there exists a path wherein future not of phi that is if there is a path say from the initial state the path of length k these are our future is bounded by k where some state is having not phi; that means, what; that means, there exists a state in the system within length k where phi is false; that means,; obviously, it is safety property does not hold in all these states so; that means, it is a counter example which you need to debug.

So, therefore, the idea of bonded model checking is that rather than checking the formula directly itself we try to find out the negation of it and then you try to find out that is the counter example and we try to work upon it. So, finding a counter example is better, because that will tell you where the bug is you can recover and the counter example basically always translates from all paths globally all paths in future to existential quantifiers. So, that you can just have to just search in 1 path; So, that is the basic goal of or idea of bounded model checking.

(Refer Slide Time: 07:43)



What is Satisfiability (SAT)

Instance: A Boolean formula.

Problem: To decide if the formula is satisfiable

In an NP-Complete Problem

$\omega_1 = (b \vee c)$

$\omega_2 = (\neg a \vee \neg d)$

$\omega_3 = (\neg b \vee d)$

$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3$

$A = \{a=0, b=1, c=0, d=1\}$

literals
clauses
SATisfying assignment!
$x+y$
$x \cdot \bar{x}$

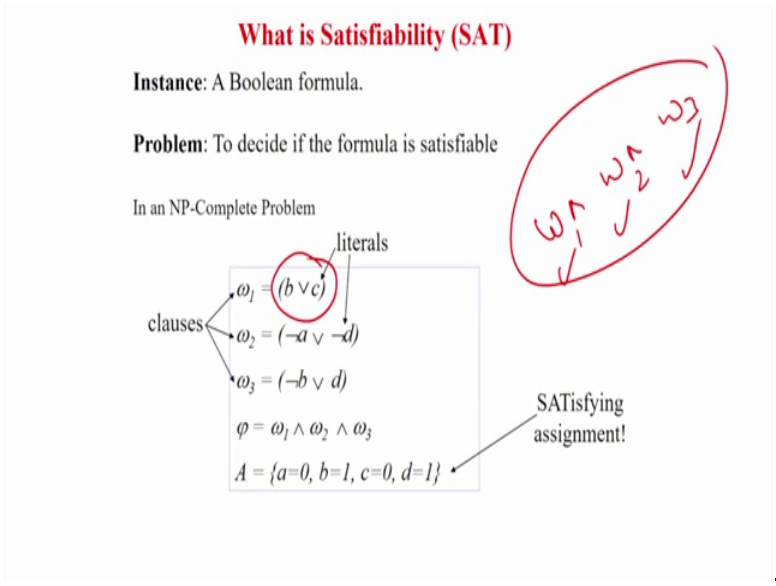
Now, which will basically see in details? So, before as and when we talk about like symbolic model checking. So, what is the idea of symbolic model checking you should know about binary decision diagrams all the labeling algorithms or 6 point computation are on the BDD itself, but when we are talking mainly about your symbolic model checking we do not use BDDS we use satisfiability of Boolean expressions or Boolean formulas.

So, just I will give you a gist what is a Boolean formula and why this satisfiable. So, you all know what is the Boolean formula, the variables are Boolean variables and then you write a function in terms of them. So, there will be and or of this Boolean or the literals as we call.

So, basically the formula where the binary a Boolean variables and all the Boolean variables or the literals basically will be having the relation of and or and basically we call it satisfiable if and only if there is some assignment on the variables, which will make the function true that is equal to 1 like. For example, a as simple as that if I say x cross y then satisfiability means x equal to 1 y equal to 1 the function is basically 1.

So, we can say that this is a satisfiable function or if you have something called x dot x bar is a unsatisfiable expression, because the whatever values here the final form answer will be are executed. So, that is there is no satisfiability value. So, this standard sat I mean if you what to read more details we can take any foundation of computer theory textbook you will be able to find out here basically, but whenever we will be talking in terms of sat we generally put it in a normal form CNF we call it a CNF form.

(Refer Slide Time: 09:09)



What is Satisfiability (SAT)

Instance: A Boolean formula.

Problem: To decide if the formula is satisfiable

In an NP-Complete Problem

literals
clauses
$\omega_1 = (b \vee c)$
$\omega_2 = (\neg a \vee \neg d)$
$\omega_3 = (\neg b \vee d)$
$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3$
$A = \{a=0, b=1, c=0, d=1\}$
SATisfying assignment!
$\omega_1 \wedge \omega_2 \wedge \omega_3$

So, in CNF form what happens we have some clauses like say omega 1, omega 2, omega 3. So, in each omega you have some of the variables we call it as literal like b or c not a and not d not b and not d. So, this will be there and all these clauses we will be basically having a and relation omega 1, omega 2. So, the whole equation to be true what should happen each of the clauses should be true; that means, this 1 should be true this 1 should be true this 1 should be true and how the clause case can be true if any one of the literals is true in that because they are or all connected by or operation.

So, if any of them is a true then literally is true in an or in a clause then the whole expression. So, if 1 literal is true in a clause then that clause is true now if all clauses are true your final formula is satisfiable. So, for example, here if I if you see that we have a satisfiable values a equal to 0, b equal to 1, c equal to 0, and d equal to 1. So, let us put it a equal 2 b equal to 1. So, b equal to 1 is 1. So, this becomes true now a equal to 0. So, not of a. So, this is becoming 1 this literally is true then basically we have d equal to 1. So, d equal to 1 means by virtue of this is equal to true. So, as all the clauses are true the formula is satisfiable. So, we call it as a satisfying assignment or the expression is satisfiable.

One thing I have to tell you at this point this problem is known to be NP-Complete. So, I mean you cannot have nobody knows about a polynomial time or an easy algorithm to solve it, but there are lot of heuristics basically which can tell you if there is a satisfiable set of Boolean solution to that it will be told. Otherwise if there is unsatisfiable will also report that, but as a as already told they are heuristics.

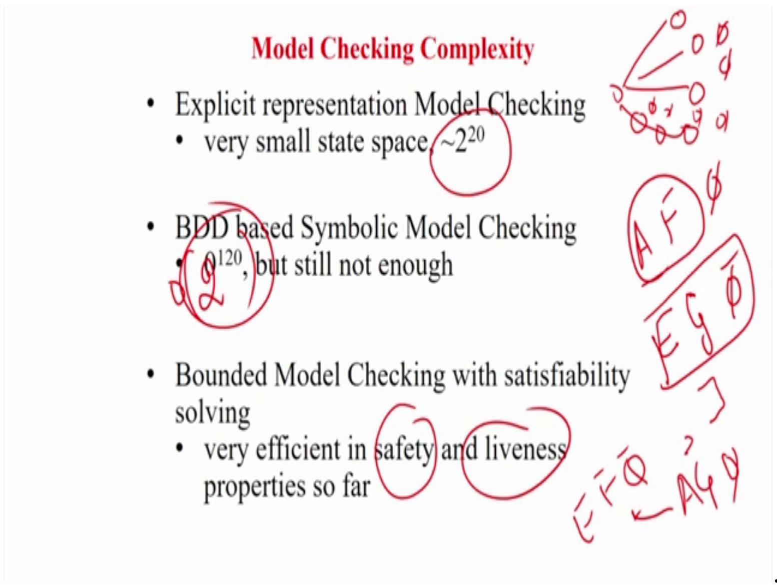
So, what are the disadvantage of heuristics will always be there they will give you the solution very quickly, but that may not be correct; that means, in many of the stray cases even if it is satisfiable the heuristic will say that is not satisfiable that may basically happen, but as there is an NP-Complete problem if you are going for exhausted enumeration based solution the time taking time taken to solve a sat will be very very large.

So, there are lot of heuristics available to solve a to find out whether a Boolean formula is satisfiable, but. In fact, there is actually entire full series of lecture should be there to know about heuristics for satisfiability, just like we had an entire series of lectures on binary decision diagram, how binary decision diagrams can be used for model checking all these things you have done in details, but in case of satisfiability we will just restrict to what is a satisfiability problem.

How verification problem or bounded model checking can be model in terms of sat and then we need to give you directions of how to solve satisfiability problem using a some exhausted algorithms, and then we will give reference no people will be telling about the heuristics, because there are all lots of heuristics to solve it.

So, this heuristics for satisfiability is not a part of the course, but rather we will try to show you how sat can be used for model checking rather the bounded model checking, but you have given a sat in normal form CNF basically we call it how people can solve it using heuristics you can read up with the basically the references given in this course.

(Refer Slide Time: 12:06)



Model Checking Complexity

- Explicit representation Model Checking
  - very small state space ~2^{20}
- BDD based Symbolic Model Checking
  - 2^{120}, but still not enough
- Bounded Model Checking with satisfiability solving
  - very efficient in safety and liveness properties so far

AF φ
EG φ
EF Q
AG φ

So, as I was telling I was just depicted in this slide. So, you can see that explicit model checking 2 to the power is the maximum size I rather call it even less that is by explicit state modeling. So, it is size can be very very less 2 to the power 10 even I sometimes say that even 2 to the power 15 is also sometimes very large to do this BDD or symbolic model checking is 1 to 2 to the power 120.

So, we rounded up 2 to the power 120 that is quite high, but still not as high as I should tell you should be able to verify noc and soc is in a as a whole. So, what people do try to do people modularize and go for distributed verification and several other distributed techniques are available or somebody goes for that if I will in explicitly verify all the blocks and then try to see the interconnect interfaces etcetera.

So, there is another set of literature to see how they can handle such big systems, but another direction which you have seen in this word people are try to model the very big systems in terms of basically other type of decision diagrams, like high level decision diagrams, or arithmetic decision diagrams and try to formally verify on both high level BDDS you can tell.

But again in as I again repeat the that a model checking as smooth or as available for a bit level BDDS are not available in this in case of those high level diagrams. So, still people are thinking about it. So, at present BDD can give you a model checking input in the order of 2 to the power 100 bounded model checking is slightly different. It can give you the higher set you can verify larger more larger systems you can verify, but mainly people are trying for safety and liveness properties. These are 2 properties which are people have been able to verify in bounded model checking with targeting very large systems, because safety as I told you just try to verify whether some formula holds in all this states.

And then you negate it and then you are trying to find out whether there exists a path where in future that form formula of the sat which is for safety is false. Liveness basically it tells the all path in future something happens; that means, say I can tell you that in all paths in future this will happen an example is I send a request in future the request will be granted. So, this is some kind of a liveness property. So, what you do receiving all paths in future some property will hold, but here as I told you the future is actually restricted by a length, but still you have to verify in all the times.

So, what people try to do they are try to found out a counter example. So, what is a counter example there exists a path where globally where globally actually maybe you can call it phi and it will be phi bar not of phi. So, that means, what if there exists a path where all the states phi is false; that means, there is there is there term is basically if I say that there exists a path where globally phi is false; that means, phi is not true at any place then; obviously, there will exists a path wherein future phi is not going to hold because; obviously, it says that is a path in where all states if you see in future this phi will be holded.

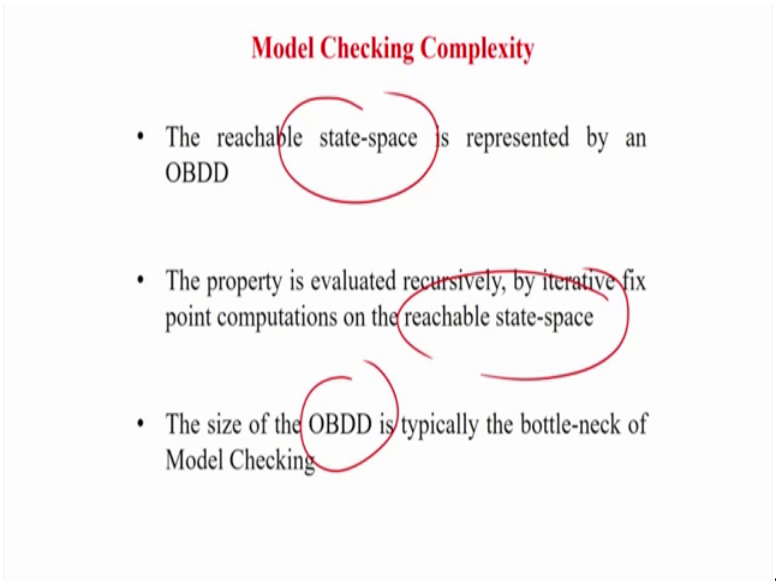
But I can find out that there exists a path like this where in all the states is not phi in all the states it is not phi. So, of course, this not phi trace will actually may I mean say that it is a the liveness is violated. So, that means, this stresses giving the counter example for liveness, but again as I told you we are always restricted by (Refer Time: 15:11) yeah. So, whenever we say that globally or in future we will just search our limit by k path length. So, that means, actually it will have the problem on handle larger systems because we are truncating or cutting down on the length to be searched.

So, basically as I showed both safety and liveness can be extend in terms for counter example where from all paths you can find out there exists a path. So, we will be searching only in some existential paths and also that also with the limited length. So, therefore so, far safety and liveness are very well studied in bounded model checking or are if you want to apply bounded model checking safety and liveness properties can be verified with very low complexity even for larger systems. For other properties generally means there is lot of other ways to handle them.

So, for the time being it is easy to understand and very obvious as I have given you that bounded model checking works on a counter example. Like all paths in future is a liveness property we will go for the negation we will find out that if there exists a path where globally this property not true; that means, you are going to find out a counter example safety means in all paths, globally some safety property should hold here we are trying to do is that we will find out that if there exists a path wherein future not phi. So, in that case basically there is a path varying future this phi is going to be a false. So, safety will be violated there. So, that is a counter example

But everywhere in future is restricted by a limit. So, therefore, your complexity is handled.

(Refer Slide Time: 16:32)



Model Checking Complexity

- The reachable state-space is represented by an OBDD
- The property is evaluated recursively, by iterative fix point computations on the reachable state-space
- The size of the OBDD is typically the bottle-neck of Model Checking

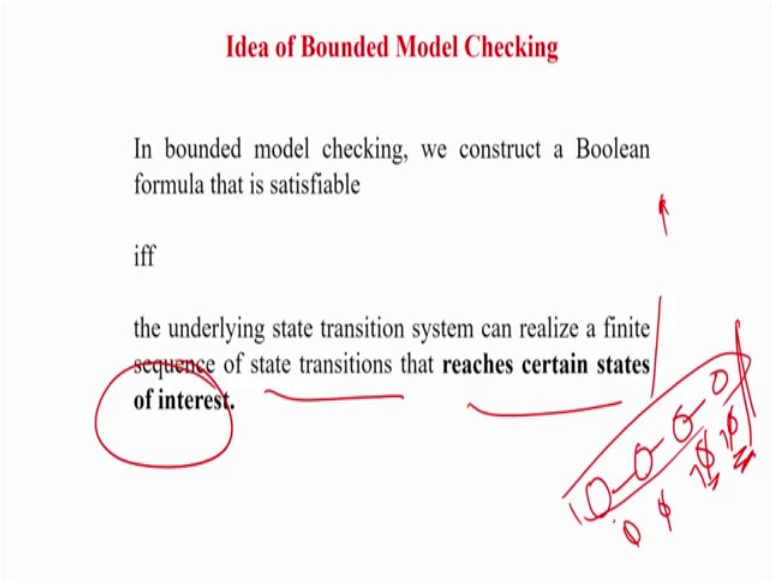
So, that is what is the idea? So, in that way bounded model checking works very well for safety and liveness property. So, basically this is just a repetition of what I am saying. So, the main problem of BDD is a reachable state space, which is quite large. So, in case of bounded state model checking we restrict the reachable state space.

So, in a OBDD or symbolic model checking the properties evaluated recursively by a fixed point computations on the reachable state space. So, in BDD the reachable state space in the entire system, but in case of bounded model checking the reachable state space is restricted by the bound. So, it actually helps us in solving it.

So, in bounded model checking; obviously, the OBDD modeling of the system is the problem. So, because the we are modeling the entire system by restricting the number of by removing redundant variables. So, it is a better length form explicit state space modeling, but still OBDD models the entire state space advantage is entire completeness and characteristics is guaranteed, but in model bounded model checking we basically restrict the state space search by some bound k. So, it actually gives a solution to a level, because as I told you we expect bugs to happen within a limit rather than postponing, it rather than think that the bugs will happen much later and not appearing earlier that is a very rare chances.

So, we take that assumption on the liberty and try to give you a very low complex solution to handle larger systems. So, that is the tradeoff basically.

(Refer Slide Time: 17:51)



Idea of Bounded Model Checking

In bounded model checking, we construct a Boolean formula that is satisfiable

iff

the underlying state transition system can realize a finite sequence of state transitions that **reaches certain states of interest.**

So, the idea bounded model checking as written formally in terms, in bounded model checking we construct a Boolean formula that is satisfiable that is the sat we have already told if and only if the underlying state transition system can realize a finite sequence of state transition that reaches certain states of interest.

That means, what we are bounded model checking means 1 side there will be a Boolean formula, which will be actually modeling this one. So, that is why there is a if and only if condition is there.

So, for each of the system you are modeling what you are trying to verify we are trying to verify, whether a sequence of state transitions. Generally in this case we try to find out existential path there is 1 path of state transitions, which reaches certain states of interest; that means, where the phi is not true phi is false that is the states where basically you want to look for the formula under question.

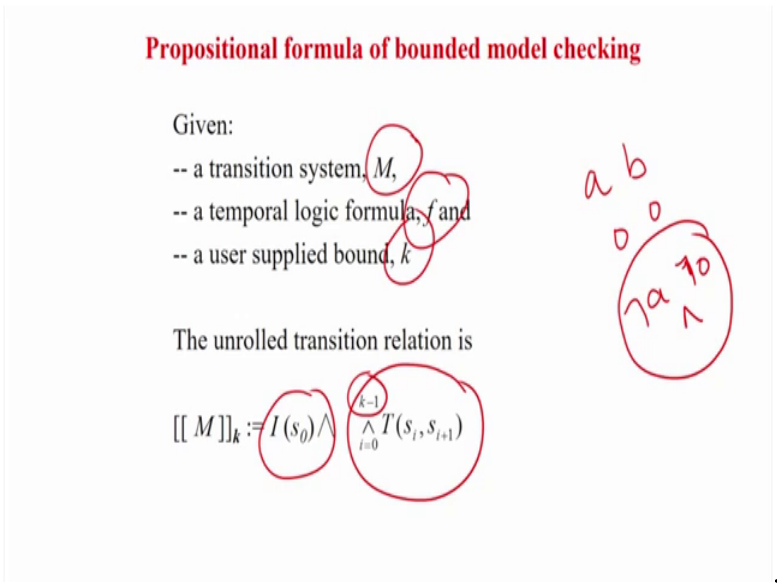
So, that is given a system we are not going to model the entire state space or reachable state space we will rather restrict it and then try to model it in terms of Boolean formula, such that if the Boolean formula is satisfy if and only if you are able to reach that search space or states restricted state space that interested formula in p channel. That means, maybe I am having signal trace just assume that I am taking an existential path someway that we are trying to verify liveness for your counter example, may be I want to find out that here this phi rather not of phi whatever you want to verify basically holds in this.

So, I will try to write this expression the transition relation and the formula phi, because I just need to note that there is a transition relation and basically in all these states what is the formula that happens. Maybe here maybe phi maybe true here, phi maybe true here, phi maybe false here also phi maybe false or some other formulas may hold or not holding any of these states. That we will try to represent in terms of Boolean formula and also the state transition relation also will write in terms of Boolean formula.

Now basically what bounded model checking will be the formula is written in such a fashion. So, that if basically you can reach a state within the bound where basically you wants to need to verify phi or not phi that is of interest. If you can reach that state with the required formula in place then only we Boolean formula will be satisfied and vice versa. So, that is basically the idea of bounded model checking.

State space path of interest you would try to express in terms of Boolean formula and such that if you are able to reach a state, which is of interest to you then and then only the Boolean formula will be satisfied and vice versa that is what is the idea? Now, with example things will be more clear to you, but because this is a very abstract statement. So, what do you do there is a propositional formula and bounded model checking.

(Refer Slide Time: 20:21)



Propositional formula of bounded model checking

Given:

-- a transition system, M,

-- a temporal logic formula, f and

-- a user supplied bound, k

The unrolled transition relation is

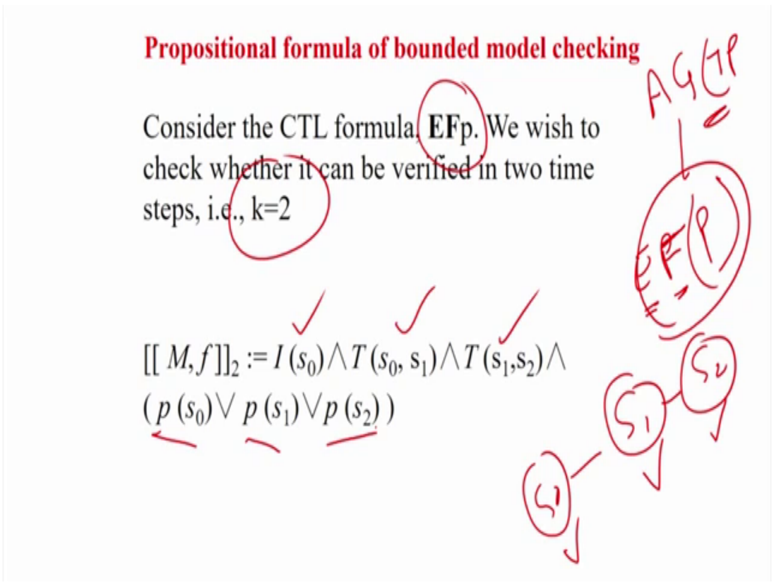
$$[[M]]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

.

Basically we have a state transition system that is your system, we have a temporal logic formula with some this propositional formula, which actually detected by temporal logic formula, which you want to verify. Like in safety as I told you given an example microwave is on and basically your door is closed. And there is a bound that is very very important that is you are going to search to a bound then we actually model the relation in terms of Boolean formula. How you do I naught s this is because the of the is the characteristics function of the initial state very simple maybe if the if we have 2 variables a and b. So, there can be 4 states 0 0 0 1 1 0 1 and so, maybe we can say that 0 0 that is not a and not b.

So, the this is the characteristic function of the initial state 0 0 and then basically we are having a transition relationship that is from Si to Si plus 1, how we are moving around it? So, this is actually going to give you the unrolled relationship, but how long you will go you go from 0 to k minus 1. The transition relation because we only maintaining up to path length of k, that is the search you are doing. So, that is basically the way we are taking a system and modeling in terms of Boolean formula with examples it will be more clear.

(Refer Slide Time: 21:26)



Propositional formula of bounded model checking

Consider the CTL formula **EF**p. We wish to check whether it can be verified in two time steps, i.e., k=2

$[[M,f]]_2 := I(s_0) \wedge T(s_0, s_1) \wedge T(s_1,s_2) \wedge (p(s_0) \vee p(s_1) \vee p(s_2))$

AG(¬p
EF(p)
s_1
s_2
s_1

So, we are having a CTL formula there exists a path where in future some p sorry wherein future F p; that means, basically it has coming it has come from the negation of some safety property like in safety what we have maybe we are having there all parts globally assuming that not p p is a state.

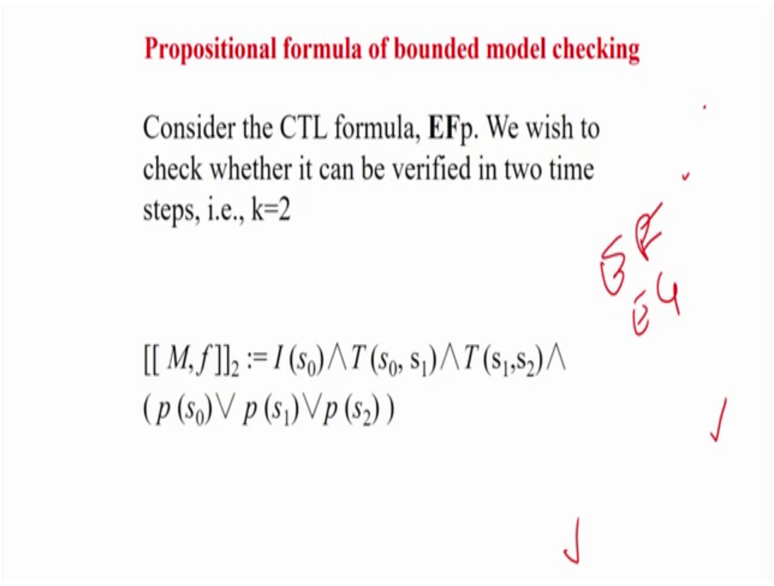
We assuming that not p is a formula which is required to be satisfied some properties negation should be there everywhere. Like for example, formula is that heater is on and door is open that should not hold anywhere basically. So, heating on an microwave door open should not be holding anywhere so, maybe all paths globally not of p. So, that is the thing you have to satisfy for global understanding global characters of the system, but again as I told you searching the entire space and writing in that is Boolean formula it will make it very large if you are having you want to quantify every path.

We just try to see the negation only; that means, if there exists a path p where in future p is there; that means, that is the negation of the safety formula there exists at least a path wherein future p is going to be true; that means, if this holds this is never going to hold; that means, basically this is the counter example; that means, there is a path wherein future at some point of time basically you know safety property is violating. So, you are going to interested only in that path because that is actually your counter example. So, that is why we call it E F p is basically a counter example, which is really want to verify in bounded model checking for safety properties how do you write it here k equal to 2 mind it. So, you are searching 2.

So, I 0 S is basically your characteristic function of the initial state, then the transition relation S 0 to S 1 S 1 to S 2 that is the 2 transition relationship, because k equal to 2. So, this part will represent the transition relationship and very important is that because you are having S 0, S 1, and S 2 this is 1 path right. And then basically so, this first 2 clauses basically is telling you about the initial state and the path length, but now if you see p S 0 p S 1 and p S 2, basically I am trying to find out that some property p it can hold either in S 0 or in S 1 or in S 2, because in future. So, a any state if it holds your job is done basically.

So, that is what I am trying to find out the p of S p of S 1 p of S 3 that is whether p holds in this state or this state or this state. So, the third clause basically is going to represent the temporal formulas truth in each of the state. So, therefore, initial state should hold the transition relation should hold in the Boolean formula from assignments and basically any one of the states should have the truth of p in this case you can then verify this formula is true or a counter example has been found. So, this is the idea we try to take a CTL formula like E f or E g and then we try to basically, as I told you we will be mainly talking about e of p and e of f and E of G because their counter examples for safety and liveness property.

(Refer Slide Time: 24:14)



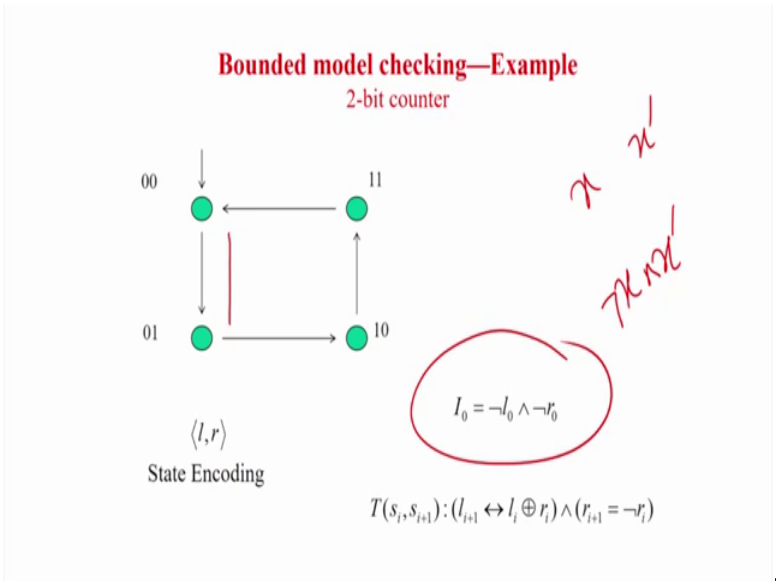
Propositional formula of bounded model checking

Consider the CTL formula, **EFp**. We wish to check whether it can be verified in two time steps, i.e., k=2

$$[[M,f]]_2 := I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge (p(s_0) \vee p(s_1) \vee p(s_2))$$

So, we can say that here in this case we are trying to see that in any of because it is in future. So, you are trying to see in any of these states the formula is to be check temporal formula is valid. So, in this case assume here satisfiable expression.

(Refer Slide Time: 24:32)



Bounded model checking—Example

2-bit counter

$\langle l, r \rangle$

State Encoding

$I_0 = \neg l_0 \wedge \neg r_0$

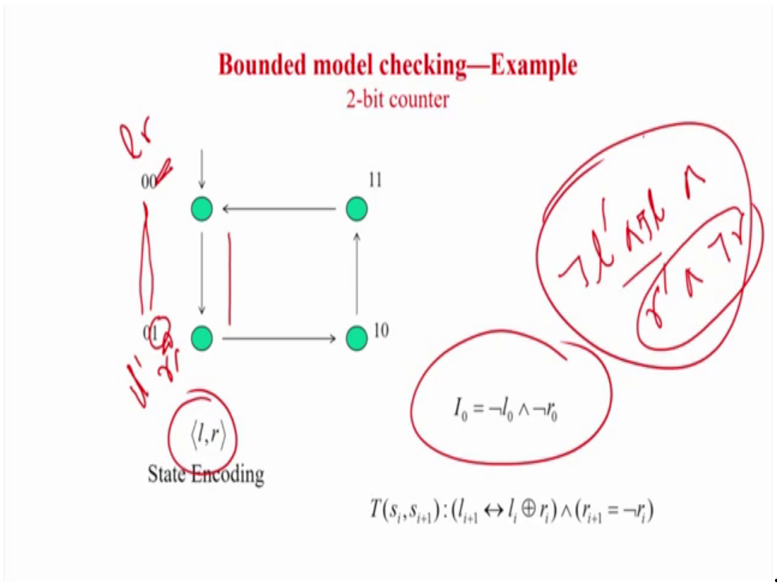
$T(s_i, s_{i+1}) : (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} = \neg r_i)$

.

Theory better to take an example, because always I means from my understanding actually if you compare testing and verification is a more mathematical problem or more because more of theoretical concepts in computer science. So, always better to explain with example simple 2 bit counter as you can see 0 0 0 1 1 0 1 1 state encoding topple is l and r.

So, initial state l naught and r 0 done, now I have to write the transition relation. So, as a as a you might have you can seen in your symbolic model checking we had something called x and x prime. So, x was this state variable corresponding the present state and x prime was a state variable corresponding to the future state. So, if you have say for example, if I want to model 0 0 to 0 1. So, you would write x not x and x prime so; that means, you are going from 0 to you are going to write it as sorry and as you are turning in terms of l and l prime. So, maybe l is the next 1 so, 0 to0.

(Refer Slide Time: 25:27)



Bounded model checking—Example

2-bit counter

$I_0 = \neg l_0 \wedge \neg r_0$

$T(s_i, s_{i+1}) : (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} = \neg r_i)$

$\neg l' \wedge \neg l \wedge r' \wedge \neg r$

So, it is will be not of l prime and l prime and not of l. So, we are using this 1 to this negate symbol we are usually represent is the negation of a variable, because prime means used to generally represent the next state.

So, this was the this was I already seen in the symbolic model checking. So, we will write it as not l not of l prime and not of l. So, not of l so; that means, which will be basically representing from 0 to 0 this is an st and also you are going to have see this 1 this 1 is one. So, basically it will be r prime and not of r. So, basically l not of l and l means this coming from 0 to 0 and because this 1 you are considering as l prime and r prime next state and this is l and r. So, basically and this 1 is from r prime from r prime to not of r. So, not of r is this 1 and r prime is basically 1 correspond to.

So, this transition can be some captured in this form. So, this is we have seen in symbolic model checking, but in case of bounded model checking we will be using this 1 as well as another way of representing, which I am going to see just concentrate on this 1 help you to represent the formulas in a smaller or a compact fashion. So, you look at it is very interesting you can see that this lsb is 0 if and only if previous both of them are xor is to be 0 like in this both of both are 0. So, 0 xor 0 is 0. So, you are going to get a 0 this 1 if you take 0 and 1 xoring is a 1. So, you are going to get a 1, now 0 and 1 if you are xoring it I a one. So, the msb is a 1.

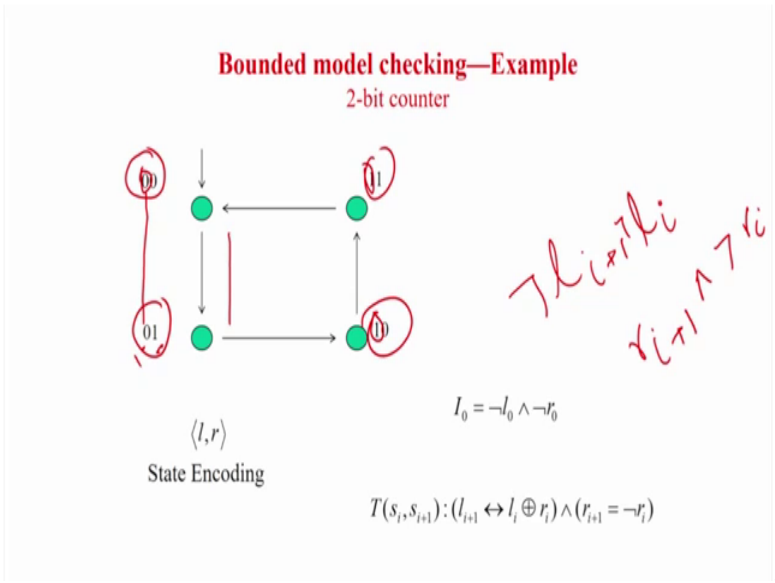
Now, 1 1 xor is a 0. So, you can you get the msb as a 0. So, msb can be very easily represented by the fact like l plus 1 l l plus 1 if and only if lix xor ri. So, in this case in the case of symbolic model checking we were using prime notation, the next state was represented by prime, but here in this case the same thing is represented by i plus 1 i is the previous state i plus 1 is the next state, because in case of bounded model checking you will have 1 2 3 4 5 6 7 8 9 10.

But, in case of symbolic model checking we only have present state next state present state next state. So, x prime and x x prime and x y prime and y would survive, but here as in a length and the length is bounded by k. So, we are using the terms li plus 1 li plus 2 li plus 3 like that instead of we will having a prime notation along with that, but that is not the issue we can have a prime notation and non prime notation because prime actually implies li plus 1 and non prime implies li.

But if it is a chain then actually you cannot you should have l l prime l double prime l triple prime like that to avoid it we are putting a lower index, but other thing is that basically we are writing it in terms of basically if and only if. So, we can also use this relation like characteristic function is l 0 r 0 that is both of them are 0.

So, as you we can also use this way of writing this states like li plus 1 and li plus means

(Refer Slide Time: 28:23)



Bounded model checking—Example

2-bit counter

$\langle l,r \rangle$

State Encoding

$I_0 = \neg l_0 \wedge \neg r_0$

$T(s_i,s_{i+1}) : (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} = \neg r_i)$

.

We can write in symbolic model checking manner also, like li plus 1 in this case not and then li also is not, because both of them are 0 and also we can write it in terms of like this 1 as this value is r. So, it is ri plus 1 that is equal to 1 over here and not of rr. So, this is all you can also represent this way instead of prime and (Refer Time: 28:48) you can put I as the indices, but there is another way of writing which I am showing over here because there will be more compact, because in case of if you want to write it in this way as I have showed for symbolic model checking then you have to model this all the 4 you have to model in that fashion.

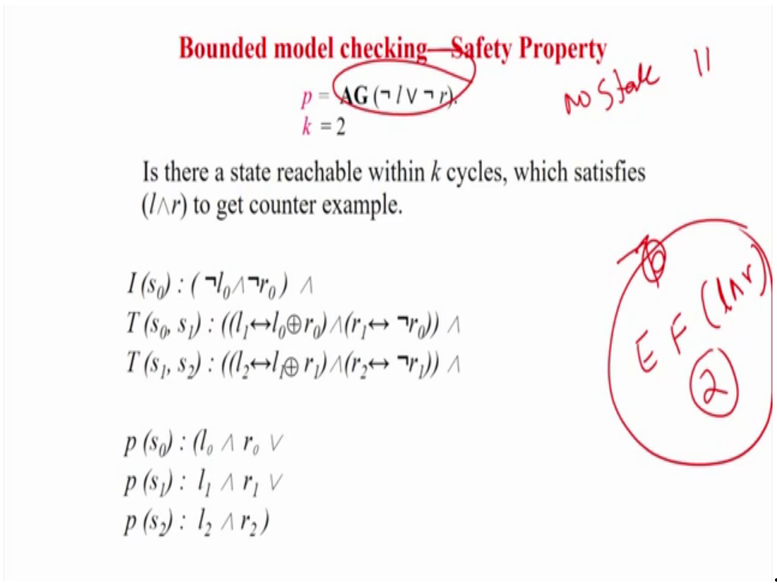
Another way of doing it is that the observation I have showed you that the msb basically is 1 if and only if the previous state a bits are x or to be 1.

But and the lsb if you see they are toggling 0 1 0 1 0 1 and 1 0. So, this 1 is 0 1, then from 1 to again 0 from 0 to this is toggling. So, therefore, another way of writing this expression both way are similar the way of symbolic model checking as I have told you or in this one. So, it is say rr i plus 1 is equal to ri that is a generic expression that if you negate the lsb. So, you are going to get the next state r next state value and the msb your whatever the msb of the next state is actually the 2 bits li and ri of the present state and you xor it. So, this is another way of writing these a the if and only if statement. So, this is another way of writing.

So, in bounded model checking you use both the way of writing. So, that it is bits more compact to represent, but actually they are all equivalent. So, you can write either in the a symbolic model checking fashion, but instead of putting r prime r double prime r triple prime you better put the indices or you can write in the if then else formula. So, both of them are same we will see.

So, basically till now if we concentrate on forget about this way of writing the expression assume that these are valid 1 both of them are correct. So, you can see a i 0 l Q 0 r 0; that means, basically it is the characteristic function of the initial state and next state lsb is basically the xor of the 2 bits of the present state and the next state that is equal to ri is not of r ri present state; present state lsb if you xor you are going to get the lsb of the next state as we have explained.

(Refer Slide Time: 30:37)



Bounded model checking—Safety Property

$p = AG(\neg l \vee \neg r)$

$k = 2$

Is there a state reachable within k cycles, which satisfies $(l \wedge r)$ to get counter example.

$I(s_0) : (\neg l_0 \wedge \neg r_0) \wedge$

$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0)) \wedge$

$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1)) \wedge$

$p(s_0) : (l_0 \wedge r_0 \vee$

$p(s_1) : l_1 \wedge r_1 \vee$

$p(s_2) : l_2 \wedge r_2)$

Now, importantly now we are going to see that how you are going to model this model of property using bounded model checking and how we are going to do it that is very important. Now, we have to pay lot of attention, we were having something called all paths globally all paths globally some formula; what is that, what is the formula saying the formula is saying that not of l and not of r that is some formula globally this should be true that is in all paths globally either not l or not r; that means, the no state sorry the no state you can have state 1 1, if there is a state 1 1 the safety property is violated, we know the safety property is either not l or not r that is either of them should be basically 0, but both of them should be 1 is not the safe state.

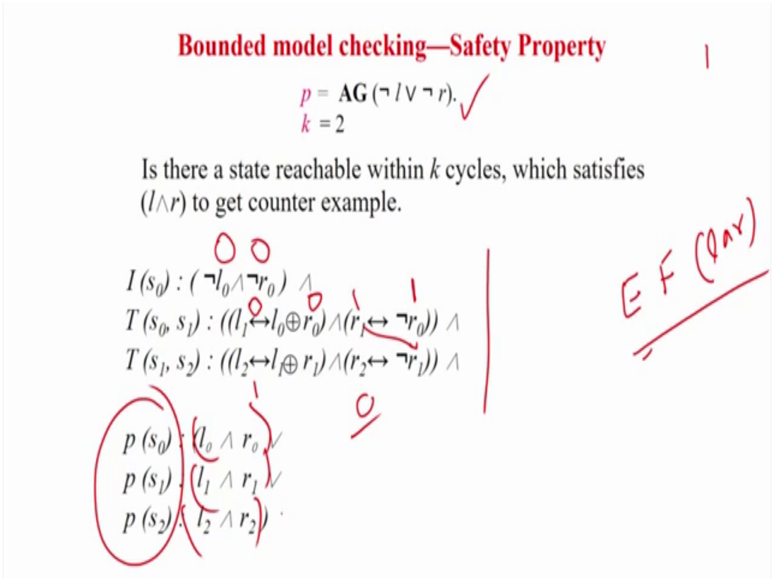
So, you have to find out that whether this property holds or not and the limit is true interestingly if you see sorry. So, only this state is having a problem, because both of them are 1 we are starting from initial state. So, bound space is only 2. So, in this case you are going to say that the formula is satisfiable, but if you make the search space 3 it will be a non-violation because it will show that no this is the state where it is violated.

So, that is the idea of bounded model checking sometimes it may give you a incorrect answer because here in this state it is not valid, but we are assuming that we can search up to 2 and our job will be done, because as I told you there we are assuming that the bugs not appear very long.

So, in this case we are keeping the bound as 2. So, of course, we are going to get a incorrect example, but then what we are going to write we are going to try to find if there is a state there exists a path wherein future many that is l and r. So, that is what is the problem both of them 1 1. So, if there is a state wherein future if this is going to be true bound is again 2 then actually we are having a case where the safety property is violated. So, you are not going to search this basically you are going to model and search this and if you find this satisfiable, then you know that the liveness property sorry the safety property is not true and there is a counter example.

So, the counter example is basically this state, but we are not going to find out in this example because you are going to take the case of 2, then we see if you make a 3 there are problem will be solved correctly, then you will be able to find out that there is a state where this safety property is violated anywhere.

(Refer Slide Time: 32:57)



Bounded model checking—Safety Property

$p = \mathbf{AG}(\neg l \vee \neg r)$. ✓
$k = 2$

Is there a state reachable within k cycles, which satisfies $(l \wedge r)$ to get counter example.

$I(s_0) : (\neg l_0 \wedge \neg r_0) \wedge$
$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0)) \wedge$
$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1)) \wedge$

$p(s_0) : (l_0 \wedge r_0) \vee$
$p(s_1) : (l_1 \wedge r_1) \vee$
$p(s_2) : (l_2 \wedge r_2)$

So, basically now, we are going to do is that we are going to find there exists a path wherein future l and r l and both of them are 1. So, how do you write it? So, if you look at it again pay attention S 0. So, basically if you look at this is the not l 0 and r 0; that means, it is basically your characteristic function of the initial state transition 1 and state transition 2 from S 0 to S 1 S 1 to S 2.

So, l 1 is nothing, but the x or of l 0 r 0 and r 1 is nothing, but not of r r 0 is not of r 1 already we have seen this in the last slide similarly for the second transition we can model it. Now, you see so, basically what we have done in this first clause this first set basically the first 1 anyway is initial state the others 2 basically models this transition and then this transition next what. So, these are the 2 transitions modeling this considering this 2 expressions t transition relationship of this 2 transitions we have shown.

Now you are just need to see that whether both of them l 0 1 1 l 1 l 0 r 0 both of them 1 are they both the of them 1 in state S 0 this formula is going to capture the fact l 1 and r 1 are they 1 1 in state S 1 this formula is capturing that, and S 2 means l 2 and r 2 are they 1 1 in state S 2. So, this 1 is actually going to capture the fact that whether the safety property is violated or not.

Now, how to find out satisfiability so, we are going to see whether it is satisfiable or not if it is free we already know it will be satisfiable, but in this 2 it will not be satisfiable and it is giving a wrong answer. So, anyway let us not at the present think that is a wrong answer because we can visualize this, but in formal sense nobody can this is a long systems so, nobody may be able to visually manually.

So, we will assuming that in this case it will be true that is no state is having the value 1 1. So, the global property that this property in all paths globally not of l or not of r is satisfiable; that means, we will not be able to find out any counter example; not finding any counter example will imply that this formula is basically true. So, let us try let us try to find out some, because they all are Boolean variables we have to find out some satisfiable operation.

So, of course, an everything has to be individually true this has to be true this has to be true and this for formula you can find out. Now let us try to find out 1 1 so; obviously, if this has to be true there is a only 1 way you can do l 0 and r 0 should be 1 1, otherwise this clause will go away fine. So, if I put l 0 and 0 0 then of course, I have to put l 0 r 0 00. So, in this case this 1 is not going to be true; that means, basically we have to now relay on l 1 r 1 and l 2 r 2. So, then this is mandatory to put 0 0 to make it true. So, I have to make it a 0 0 here.

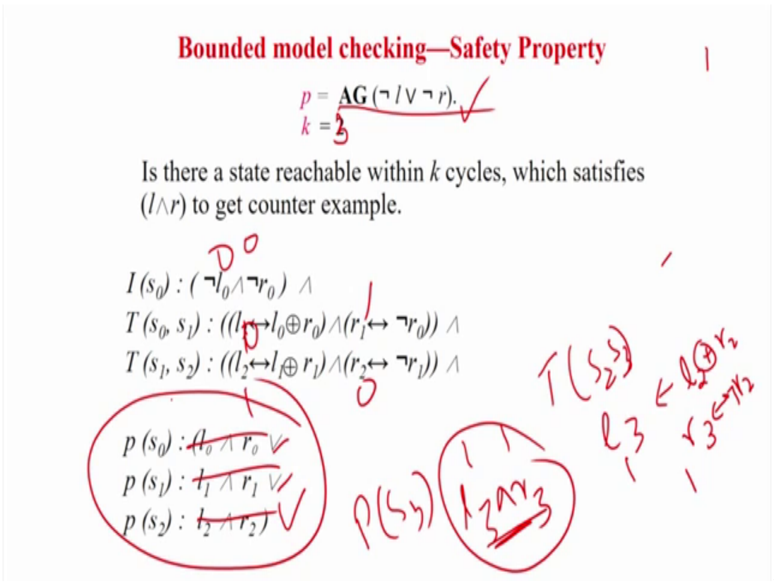
Now, you can also find out appreciate the you will be able to appreciate the fact that basically this 1 is already 0 l 0 is l 0 is 0 r 0 equal to 0. So, this is also equal to 0. In fact, that means, equal to 1 negate it and this 1 l 0 r 0 means basically it is a 0; that means, l 1 equal to you have to fix this because not l 0 and not r 0 means both of them has to be 0 0 means l naught has to be 0 r 0 has to be 0. So, that this clause becomes true.

So, that is fixed; now if you fixed this when this 1 is becoming 1 and l 0 r 0 becoming 0. So, l 0 is equal to l 1 is equal to 0. So, l 1 will be equal to 0 and r 1 will be equal to a 1. So, even this 1 is also not satisfying not becoming 1 gone, now only 1 thing is remaining the last 1. So, if you calculate this 1 will be 1 and this 1 will be equal to a 0, because this r 0 is 1. So, this 1 is basically equal to 1 not of r 1 means basically it is a 0. So, 1 0 again this is 1 and this is 0.

So, basically no how no way you can actually satisfy this clause clauses together, because it is says this or this or this has to be satisfied, because this I can do if I put l naught 0 r 0 0 then assignments. I have shown you you can easily satisfy these 3 clauses, but here you know none of this 3 is s 0 is 1 and s 2 you can have the value satisfiable.

Because it is 0 0 0 1 1 0 is not satisfied. So, therefore, it will tell that it is a nonsatisfiable Boolean formula and therefore, no counter example has been found. So, your job is done. So, you can say that this is the property holds true for k equal to 2, but now as you see as you have already seen this is a wrong answer wrong answer because we have thinking a lower bound. So, what I have to do I have to increase my bound 2 3. So, in this case of course, it is an homework you can try out.

(Refer Slide Time: 37:21)



Bounded model checking—Safety Property

$p =$ **AG** $(\neg l \vee \neg r)$. ✓

$k = 2$ 3

Is there a state reachable within k cycles, which satisfies $(l \wedge r)$ to get counter example.

$I(s_0) : (\neg l_0 \wedge \neg r_0) \wedge$

$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0)) \wedge$

$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1)) \wedge$

$p(s_0) : (\cancel{l_0 \wedge r_0} \vee$

$p(s_1) : \cancel{l_1 \wedge r_1} \vee$

$p(s_2) : \cancel{l_2 \wedge r_2})$

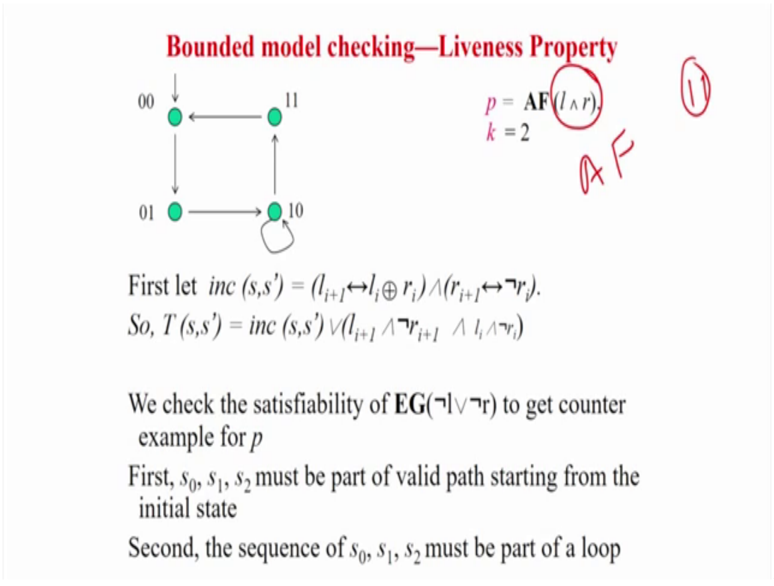
.

If there will be another expression called t of s 2 to s 3 and then this 1 will be equal to l 3 xor l 2 xor r 2 right. And then it will be r 3 equal to xor r 2 not of r 2 and this case you will have another one. So, this 1 will be or another P of S 3 will be equal to l 3 and r 3.

So, now you can easily find out that this will be satisfiable because if you put 0 0 you are going to get the answer 0 1 1 0 and the final will be this 1 will be l 3 will be l 3 will be 1 1 we are going to get it and then; obviously, this is false this is false this is false, but basically l 3 and r 3 both of them are 1.

So, this one this clause will be making because they are these are all odd if you see this is these are these are odd. So, this one will be true making this entire set true and of course, there will be a satisfiable assignment next you can calculate it in your home and you can find out. So, if I make k equal to 3. So, this clauses will change and you are going to get a satisfiable assignment with it from the counter example; that means, that is going to tell you that this path basically is a counter example and then you can debug a find the answer. So, done.

(Refer Slide Time: 38:33)


00
11
01
10
11
AF

Bounded model checking—Liveness Property

p = **AF**$(l \wedge r)$,
$k = 2$

First let $inc(s,s') = (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} \leftrightarrow \neg r_i)$.
So, $T(s,s') = inc(s,s') \vee (l_{i+1} \wedge \neg r_{i+1} \wedge l_i \wedge \neg r_i)$

We check the satisfiability of **EG**$(\neg l \vee \neg r)$ to get counter example for p

First, s_0, s_1, s_2 must be part of valid path starting from the initial state

Second, the sequence of s_0, s_1, s_2 must be part of a loop

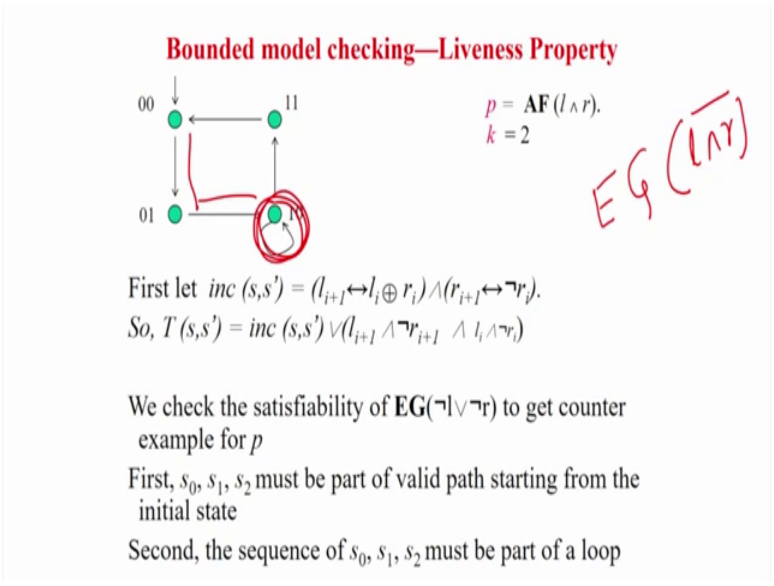
So, that was simple. So, you have we have shown with an example how basically we can handle the so, called your safety property. Now we are going to see the liveness property. So, what is the liveness property. So, one thing I should tell you that this Boolean satisfiability check I have done manually like putting 0 1 and so forth, but in the general case it will be formula in CNF form which I will discuss shortly then they are automatic solve us to do it; mainly exhausting solving is an np complete problem. So, there are heuristics. So, this road go by this notion that I have done it manually. So, it is possible to do it manually for large scale there has to be an automation and I needed. So, now, I am going to slightly change because we are now going to see the liveness property.

So, what I told you. So, liveness property means here the here they what they are requiring they are requiring in all paths in future l and r; that means, they want to say that then there should be all path if you take from the initial state in future you should have the value of 1 1 in the state.

So, if you look at these example of course, it is true if you take any path in future we are going to get this which is going to be a true, but here we are actually slightly changing the example here I will putting a loop same counter, but I am putting a loop over here. So, the formula will be false basically because I go over here I go over there will be a.

So, in all paths in future you are not going to have the value 1 1 and again bound I am taking as a 2 you can take 3 or whatever, because I am going to get the gist of it. So, now, fine done; So, now, basically how I will told you how will you solve it? So, as I told you that we will have to take there exists a path.

(Refer Slide Time: 39:56)



Bounded model checking—Liveness Property

$p = \mathbf{AF}(l \wedge r)$.
$k = 2$

$EG(\overline{l \wedge r})$

First let $inc(s,s') = (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} \leftrightarrow \neg r_i)$.
So, $T(s,s') = inc(s,s') \vee (l_{i+1} \wedge \neg r_{i+1} \wedge l_i \wedge \neg r_i)$

We check the satisfiability of $\mathbf{EG}(\neg l \vee \neg r)$ to get counter example for p

First, s_0, s_1, s_2 must be part of valid path starting from the initial state

Second, the sequence of s_0, s_1, s_2 must be part of a loop

Where globally l and r negate; that means, there should be a path where in all states 1 1 is false of course, it is there because you can go over here go over and there is a say loop.

And one thing I should tell you I am going into more details because that will make the things very complex, that when I am saying needs of size 2 length of size 2 path 2 basically this loop here I am not going to count explicitly. Because otherwise it will make problem entirely wrong because it is an infinite loop. So, your bounds will be infinite. So, to keep the thing simple I am not going into the formalities, basically see I going formal I am not going to formal analysis of this.

But just keep this gist idea or put uncode idea in mind that when we are looking for liveness 1 length 2 length after that we are going to not explicitly counting loops. Where we are trying to see that I will show you how to do that rather will try to find out, that this is the path this is the path. And in this paths basically globally this property should be false, that is 1 1 should be false. And also as liveness globally means what there should be a loop between this 3 states; that means, in these case in these state globally this should be false; that means, here also it should not be 1 1 here also it should not be 1 1 and here it should not be 1 1.

And there should be a possibility that you cannot exists exit from these 3 stage or in any one of this sate or any two of these state; that means, we are found a subset of states which are we are calling globally with respect to bound 2. So, in case what n 2 will be bound and you cannot adding all the states within that bound your property that is 1 1 is false and you should not be able to come out of this set of states. So, it can be either there can be a loop from here to here. So, that will be looping in this 3 or it can also have the case that you can loop back from here you can also have a loop over here; that means, these are the 3 states which are violating these property at this in a path.

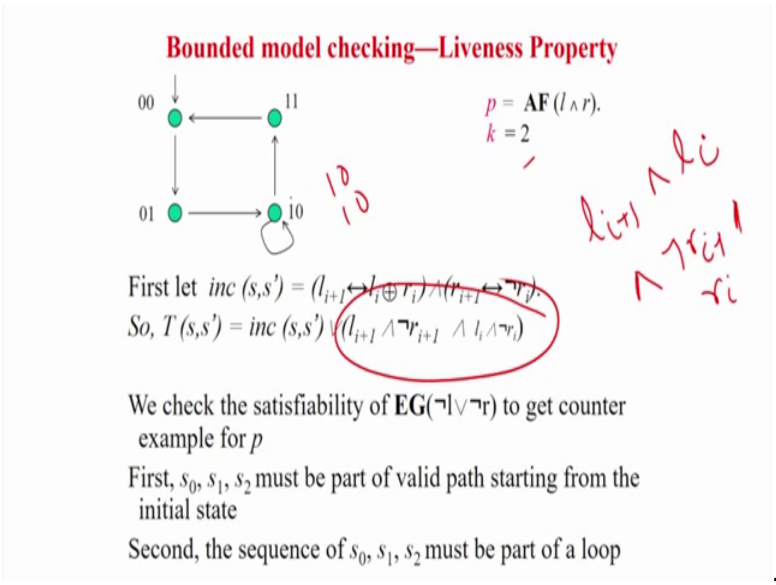
And you should never be able to come out of that path. So, there should be a loop. So, in this case you can find out the loop is in this one that is also because you will not be able to come out from this 3 states any of the 3 states where the property is not true.

So, therefore, it is true that basically in future you will be never be able to find out you will in future you will not be able to say that you will be able not be able to reach this things. So, liveness property is not valid basically, because may be may be 1 that after sometime 1 1 should reach, but in this path basically you will be looping over here and your job is gone right. So, let us see how do you write it and how do you do it. So, again

I am not here this will be similar this one will be similar the first set this one corresponds to.

Basically your flipping condition and there is only 1 loop. So, if you look at it how this only loop from 1 0? So, basically this we can write by the symbolic state transistor enumeration manner.

(Refer Slide Time: 42:40)



Bounded model checking—Liveness Property

$$\begin{tikzcd}
\arrow[d] 00 & 11 \arrow[l] \\
01 \arrow[u, phantom] \arrow[r] & 10 \arrow[u] \arrow[loop below]
\end{tikzcd}$$

$p =$ **AF** $(l \wedge r)$.
$k = 2$

10
10

$l_{i+1} \wedge l_i$
$\wedge \neg r_{i+1}$
$\wedge r_i$

First let $inc(s,s') = ~~(l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} \leftrightarrow \neg r_i)~~.$

So, $T(s,s') = inc(s,s') \vee (l_{i+1} \wedge \neg r_{i+1} \wedge l_i \wedge \neg r_i)$

We check the satisfiability of **EG**$(\neg l \vee \neg r)$ to get counter example for p

First, s_0, s_1, s_2 must be part of valid path starting from the initial state

Second, the sequence of s_0, s_1, s_2 must be part of a loop

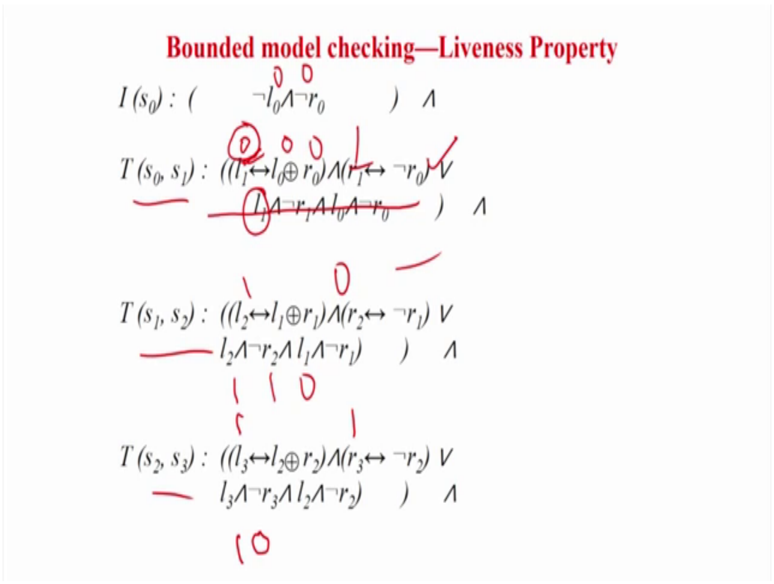
So, it can be 1 it will be li plus 1 msb and li correct because same loop it is from 1 0 to 1 0. So, li plus 1 li and ri also is same. So, you will have not of ri plus 1 negate and ri. So, this is the same thing which we have shown over here. So, this shows that li plus 1 and not of ri; that means, I have just swap the position. So, li plus 1 and li not of ri plus 1 is going to naught ri that is basically you have just modeling this loop.

So, this path we are writing just a symbolic way of enumeration of this transitions only instead of bars x bar x double bar x triple bar, basically x 2 x bar within x state present. So, we have just numerating with the what do as the I the subscript.

So, basically we are going try to try to figure this out that there exists a path where globally not this has to be true; that means, 1 1 should not be true in this path. So, we are going to find out a path called S 0 S 1 and S 2 level is 2 and very important is this one the third last statement there should be a part of the loop.

So, first one is the initial state then you will have a state transition initial already we have discussed in the previous example and as is a we are going to check globally. So, there should be a way that you cannot come out of this either from this state or this state or this state or any one of them; that means, you should not be able to exit S 0 S 1 S 2 or any one of them.

(Refer Slide Time: 44:07)



Bounded model checking—Liveness Property

$I(s_0) : (\quad \overset{0}{\neg l_0} \wedge \overset{0}{\neg r_0} \quad) \wedge$

$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0) \vee$

$\quad \cancel{l_1 \wedge \neg r_1 \wedge l_0 \wedge \neg r_0} \quad) \wedge$

$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1) \vee$

$\quad l_2 \wedge \neg r_2 \wedge l_1 \wedge \neg r_1) \quad) \wedge$

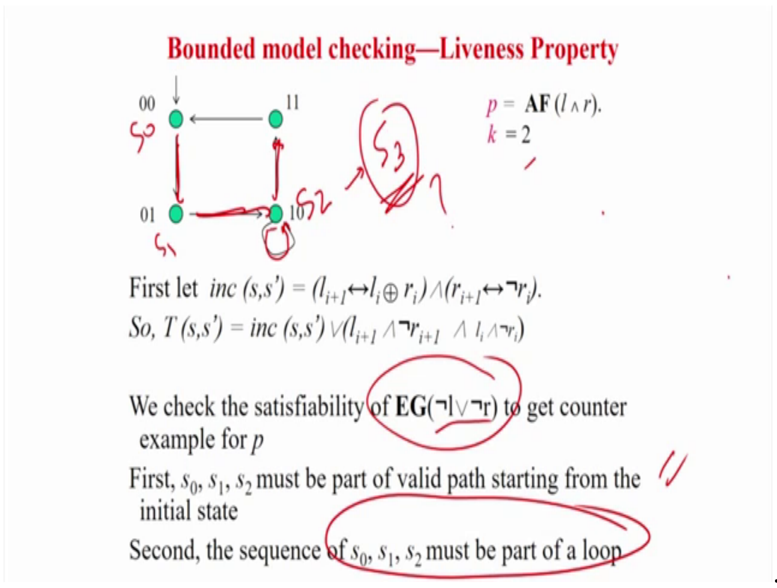
$T(s_2, s_3) : ((l_3 \leftrightarrow l_2 \oplus r_2) \wedge (r_3 \leftrightarrow \neg r_2) \vee$

$\quad l_3 \wedge \neg r_3 \wedge l_2 \wedge \neg r_2) \quad) \wedge$

So, that is the idea now we will see how to model it. So, I am not going to explain too much the this is the S 0 is basically your characteristic state of the function initial state, this is your first transition S 0 to S 1. So, this one is the xor relation for the counter and if you look at it I am adding this one l 1 and not r not r 1 l naught and r r 0, that is the self-loop, but in this case it will be false because the self-loop is not present in the initial state.

Similarly this one is from S 1 to S 2 and this 1 is from S 2 to S 3 this one is very very important you have to keep everything in mind l 3 r 2 this 1 and l 3 not of r 3 and l 2 not of r 2; that means, what we have done just keep this in mind. So, what we have done over here.

(Refer Slide Time: 44:53)


Bounded model checking—Liveness Property

00
11
01
10
S0
S1
S2
S3
?

p = **AF** $(l \wedge r)$.
$k = 2$

First let $inc(s,s') = (l_{i+1} \leftrightarrow l_i \oplus r_i) \wedge (r_{i+1} \leftrightarrow \neg r_i)$.
So, $T(s,s') = inc(s,s') \vee (l_{i+1} \wedge \neg r_{i+1} \wedge l_i \wedge \neg r_i)$

We check the satisfiability of **EG**$(\neg l \vee \neg r)$ to get counter example for p

First, s_0, s_1, s_2 must be part of valid path starting from the initial state

Second, the sequence of s_0, s_1, s_2 must be part of a loop

So, we have a transition register S 0 S 1 S 2 and S 3 basically is at present S 2 only it is a some state. I am keeping hanging just keep it in mind.

So, what are we are doing this one is the transition very simple there is 1 transition from this from S 2 to S 3 is a question of important there are 2 cases 1 can be S 2 to s 3 where S 3 is equal to S 2 that is the loop itself. And another can be this one right path this can be a self-loop as well as this can be a to be next state. So, this one I am keeping in as a question mark now what I am modeling. So, this is state 1 this is state 2 and this one is state 3 basically to model this one.

So, I mean just you as I have done manually you can easily do it if you just put it 0 0 that has to be done for satisfiability, you will just get it 0 1 in this case you will get it 1 0 in this case you are going to get 1 1 or or basically you see in this case this is not going to be satisfiable, this is not going to be satisfiable by any way, but in this case you can have 1 3 also as 1 0 that is very important, because I mean just by calculating you can find out.

So, if you take 0 0 basically, you are going to get the answer here as 1 0 so, I mean just I mean just should not tell it like that; I mean just I am I will try to take it the other way. So, 0 0 this 1 will be assign 0 1 this will be 1 0 and this 1 is actually 0 1 1 0 this is 0 1. So, the answer is the next state basically is 0 1 the next state is 1 0 here this state is 1 1.

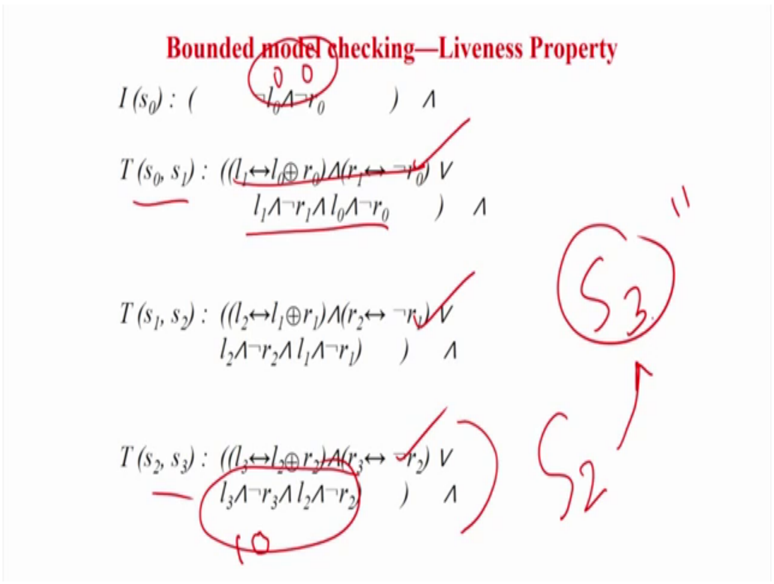
And the ground states basically this is 1 0 this one this l 1 1 l 1 1 r 1 for the this clause the top clause will lead to 0 1 if you are going by the lower clause. So, the it will be basically also 0 1, we will see how it is be sorry. So, that means, basically it is a 0 0 and if you take these values by the first clause it is basically 0 1 that is the next state, but if you are taking by this one also there is a possibility you can have this 0 0 else l 1 0 and r 1 0 both of them like where like for example, if I take l 1 is equal to 0 fine.

So, I am going to get this is the answer basically it is showing that l 1 xor this 1. So, we know that both of them are 0 0. So, if you take 0 0 the answer of l 1 is a 0. So, if you are going to get this as 0; that means, this is not going to true over here.

Similarly, in this case if you see l 2 if I take this clause. So, you are going to get l 2 equal to 1 and r 2 equal to 1 just like the simple expression and then this in this case you can see r 2 is basically equal to 0. So, this is equal to 1 l 2 is equal to 1 l 1 basically is equal to 0 from this clause. So, you can see this is 0. So, it will be 0.

So, if you just do a simple 1 calculation means just you have to do it that is why we setting up the values which I which is very boring to do explicit.

(Refer Slide Time: 48:15)



Bounded model checking—Liveness Property

$I(s_0) : (\quad \neg l_0 \wedge \neg r_0 \quad) \wedge$

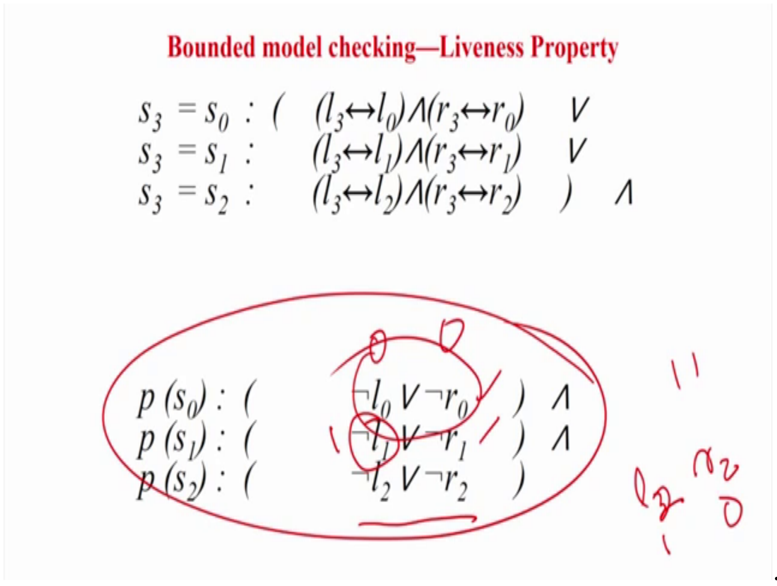
$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0) \vee$
$\quad l_1 \wedge \neg r_1 \wedge l_0 \wedge \neg r_0 \quad) \wedge$

$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1) \vee$
$\quad l_2 \wedge \neg r_2 \wedge l_1 \wedge \neg r_1) \quad) \wedge$

$T(s_2, s_3) : ((l_3 \leftrightarrow l_2 \oplus r_2) \wedge (r_3 \leftrightarrow \neg r_2) \vee$
$\quad l_3 \wedge \neg r_3 \wedge l_2 \wedge \neg r_2) \quad) \wedge$

In the lecture you will be able to find out that this has to be 0 0 this 1 will be satisfied this one will be satisfied in this case I mean this is going to be satisfied, but s 3 is basically hanging in next slide will make this whole picture whatever I was talking in a bit confusing manner will be cleared, but only thing you take in mind that all of them has to be true individually and they can be true either by virtue of this or by virtue of this. The first one goes from this state to next state and next one represents the self-loop.

(Refer Slide Time: 48:35)



Bounded model checking—Liveness Property

$$\begin{aligned}
s_3 = s_0 &: ((l_3 \leftrightarrow l_0) \wedge (r_3 \leftrightarrow r_0) \vee \\
s_3 = s_1 &: (l_3 \leftrightarrow l_1) \wedge (r_3 \leftrightarrow r_1) \vee \\
s_3 = s_2 &: (l_3 \leftrightarrow l_2) \wedge (r_3 \leftrightarrow r_2)) \wedge
\end{aligned}$$

$$\begin{aligned}
p(s_0) &: (\neg l_0 \vee \neg r_0) \wedge \\
p(s_1) &: (\neg l_1 \vee \neg r_1) \wedge \\
p(s_2) &: (\neg l_2 \vee \neg r_2)
\end{aligned}$$

0
0
1 1
1
l_2 r_2
1 0

Next, if I go it will be clear. So, these 3 last 3 already we have explained in the last case in each of this state the property you are inter stating. So, the property you are inter stating is that this should be true not l 0 not r 0; that means, no 1 it should be 1 1. So, that is; obviously, valid because if any of the states you are having 1 1.

Then of course, you see that it will one of the clause will be false and there is a and relation. So, this 1 we know that 0 0 l 0 r 0. So, if this 1 will be true and l 1 is 0 r 1 is a 1. So, this is l 1 is a 0; that means, this 1 will going to be a 1, because l 1 is 0. So, not of l 1 will be making it true and if you look at this 1 we know that l 3 is 1 and r 2 is equal to sorry l 2 is equal to 0 and r 2 is equal to 1 0 that is state S 2 l 2 1 r 2 0 so in fact, this 1 is going to be a 1 and this is true.

So, in fact, first I am writing down the values, which will actually make every state true and all them has to be true. In fact if I keep any of these values as 1 1 if you check the each of these clause will be false and we are going to get the answer as the wrong.

So, to make it satifiable we have to make r 0 l 0 r 0 0 0 l 1 is 0 r 1 is 1 and l 2 is 1 and r 2 is 0, you can easily see that by making these 2 this is satisfied and by 1 1 0 will be satisfying this one and r 2 is 0 r 2 is 0. So, it enables satisfy this one so in fact, none of the states is 1 1. So, globally everywhere not 1 1is satisfied by these 3 states so; that means, p S 0 S 1 and S 2 everywhere, we are satisfying the none of the states are the variable that is 1 1 done. So, these states are checking your properties of globally all sates the negate of the property (Refer Time: 50:22) is true.

Now, this is very important which you have to think a lot. So, what it is saying. So, initially we have a this one this one is of importance. So, this says that there is a transition from S 2 to S 3 and you have 2 2 ways of doing it 1 is this first clause, if you see it is saying that it goes from S 2 to S 3 that is basically in this case S 3 corresponds to 1 1.

Another transition and the second clause if you see it allows again S 2 to go through S 3, but in that case it is a self-loop; that means, the second clause will tell that S 3 is nothing, but S 2 there are the 2 ways of doing it, but as we have also seen as the clause for the liveness property you should verify that S 0 S 1 and S 2 you cannot leave that by means of a self-loop.

That means, what then what is S 3, then S 3 cannot be this 1. So, what are the viable changes of S 3 S 3 can be equal to S 0. So, in that case there is a transition there will be a loop S 3 can also be equal to S 1 in this case there will be a self loop like this also it can be S 2 can be S 2 itself; that means, S 2 can be S 2 itself; that means, there will be self-loop which it is a true S 2 in this case. So, you have to model it something like this that is S 3 is nothing, but S 2. In this case the transition is something like this it is something like this correct.

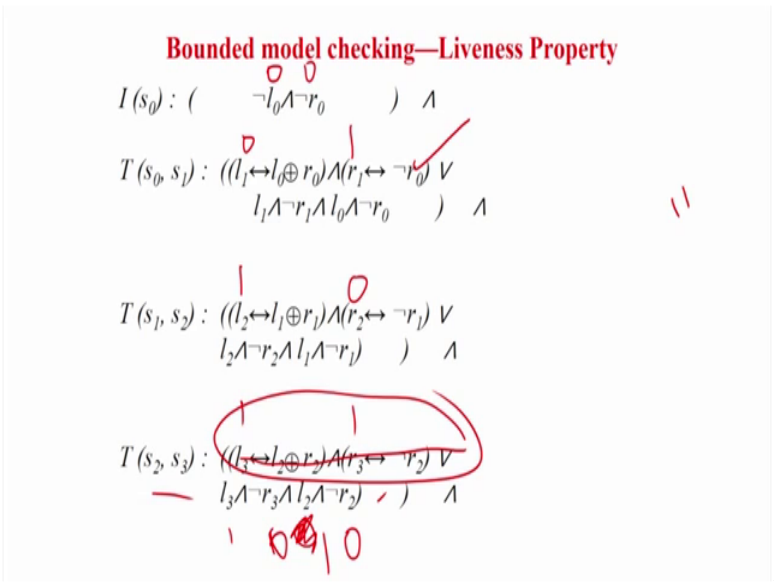
Then basically like this then in this case S 3 is equal to S 2 correct and also there may be case like self-loop. So, therefore, the first clause here the first clause here it is telling that basically S 3 is equal to S 2; that means, this is self-loop; that means, what l 0 equal to l 3 and r naught equal to r 3. So, this is modeling the fact that S 0 is equal to S 3 that is this one.

The second 1 what it is saying this one. So, how it is reflected l 1 equal to l 3; that means, l 3 is nothing, but corresponding to S 3 is equal to l 1 and r 3 equal to r 1; that means, there is a S 3 is equal to sorry this is S 0 S 1 and S 2 I am sorry this we are telling as S 3 sorry.

So, the first one is sorry the first one is telling something like this S 0 is S 3 is equal to S 0. So, this one second one is saying any one of them may be true second 1 is this 1 and third clause in the self-loop, which was actually saying that S 3 is equal to S 2.

Now, we any one of them should be do because the or you see that is or. So, in that case in this case basically it is moulding up the loop you will find out the this is going to be basically true satisfiability. So, now I will start putting the values of satisfiability you can easily check this.

(Refer Slide Time: 53:09)



Bounded model checking—Liveness Property

$I(s_0) : (\neg l_0 \wedge \neg r_0) \wedge$

$T(s_0, s_1) : ((l_1 \leftrightarrow l_0 \oplus r_0) \wedge (r_1 \leftrightarrow \neg r_0) \vee$
$l_1 \wedge \neg r_1 \wedge l_0 \wedge \neg r_0) \wedge$

$T(s_1, s_2) : ((l_2 \leftrightarrow l_1 \oplus r_1) \wedge (r_2 \leftrightarrow \neg r_1) \vee$
$l_2 \wedge \neg r_2 \wedge l_1 \wedge \neg r_1)) \wedge$

$T(s_2, s_3) : ((l_3 \leftrightarrow l_2 \oplus r_2) \wedge (r_3 \leftrightarrow \neg r_2) \vee$
$l_3 \wedge \neg r_3 \wedge l_2 \wedge \neg r_2)) \wedge$

So, you put start putting it as 0 0 this is the only way to do it. So, if you do it basically l 1 will be 0 and this is equal to be 1. In this case it will be 1 and this is equal to 0, here it will be interesting again l sorry here again l 3 basically, in this will not be satisfiable because if you take this l 1 will be equal to 1 and l 1 will be equal to 1.

So, I am taking this one I am taking that again l 3 equal to 1 and l 2 equal to 0. So, if you look at it because sorry l 3 equal to 1 and r 3 is also equal to 1 1 0 sorry 1 0 1 0. So, you can find and l 2 you already know as 1 and r 2 you already know as 0. So, this 1 0 1 0 will satisfy this, we are not targeting this. So, any one of them will hold. So, I am taking this. So, I am going to get this. So, I am please remember that I am taking l 1 l 0 r 0 0 0 will has to be done for this then l 1 r 1 is this l 2 r to is 1 0 and l 3 r 3 I am also taking as 1 0 I am not taking as 1 1 because I have a choice.

Either this can be true or this can be true, but of course, this one I cannot put a loop because it will make it false basically so, any. So, now, so, these are the values. So, in this case I am not taking this I am taking this one.

So, now, here if you look at it basically this is always true I have put and now you put the values of this one l like because what is l 3 is 1 and r 3 is 0 ok. What is l 0 l 0 is 0 r 0 is 0. So, this clause is not verified to be true not satisfied by this 1, but any one of them will hold. So, now, if I put it as l 3 as 1 r 3 as 0 so, l 1 is 1, but r 1 is 1. So, again not verified here if you see 1 1 1 0 this is also l 1 is 1 r 2 is 1. So, this equivalence is done and this one is true.

So, the whole formula is satisfiable; slightly tricky think read the notes and work it by hand. So, just your nutshell what is what is the values and there are actually said this will be written as I will show you in the next slide that, how you can write basically all the formulas I am showing you here; like this one is already a CNF, but if this provisions basically this if and only if how do you write it in terms of simple Boolean and or operation I will show you in the next slide.

But basically what happened just satisfiable BDD will start with 0 0, then this I will be satisfiable because anyone or is there anyone of these three this I will do it. In this case I will not do this I will take this to be satisfiable and then of course, we will find out that here is again or of and. So, this 1 will be not this 1 will be not, but this will be true and this case everyone is satisfiable because this is a and operation.

So, it is actually shows that this is satisfiable; that means, we have generate at the counter example here very loops and basically yeah not reaching a state of 1 0 within a bounded time and you are basically generate your counter example; that means, you are saying that in all paths globally you will finally, reach this state 1 1 it is basically false because here we are going to have when you get just stuck up and here it is a counter example.

So, this is your basically your counter example. So, now, what I have to do you have to basically eliminate this. So, if you are going to eliminate this means this will be going out. So, in this case this one will be going out, this one will be going out, this 1 will be going out and of course, anyway you have to keep this check on and then you can study the satisfiablity, but before that you try to make k equal to 3 because in 3 steps we will be reaching this.

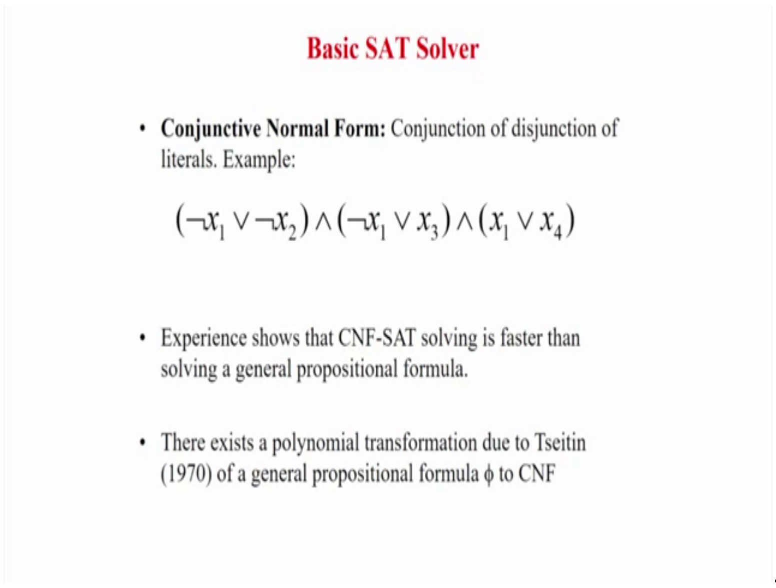
So, try to if you want to solve it or you have to find out that globally the property is true or not then basically you have to eliminate this debugging this is the debugging, because the self-loop is killing you. So, you make do this and tend to find out whether this is true or not, but again you will be finding out the problem because length is this 1 make it 3 and to something you can do it and you will be getting the answer. Again slightly complicated read it and do it 1 by 1 the values will be you will be able to find out now what we means that whatever still.

Now, we have being doing we have all solving it by means of what we are solving it in by means of you have to model we are modelling basically the some temporal formula and basically we are writing it in terms of some Boolean expressions. And these Boolean expressions basically we are trying to find out whether they are satisfiable or not.

So, these are this is already in the in kind of a and or Boolean and or, but this is slightly if and only I have convert it in to and or if I will do then you will have a Boolean formula and then there are satisfiability solvers, which will tell you whether there exists any Boolean the variable value with make it is true if that is there then that that values you have to study and if it is unsatisfiable; that means, your formula you are going to verify is false.

And there you have to act accordingly. So, that is what now remains to be told that what is SAT? And how basically it solves this problem?

(Refer Slide Time: 57:57)



Basic SAT Solver

- **Conjunctive Normal Form:** Conjunction of disjunction of literals. Example:

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee x_4)$$

- Experience shows that CNF-SAT solving is faster than solving a general propositional formula.
- There exists a polynomial transformation due to Tseitin (1970) of a general propositional formula ϕ to CNF

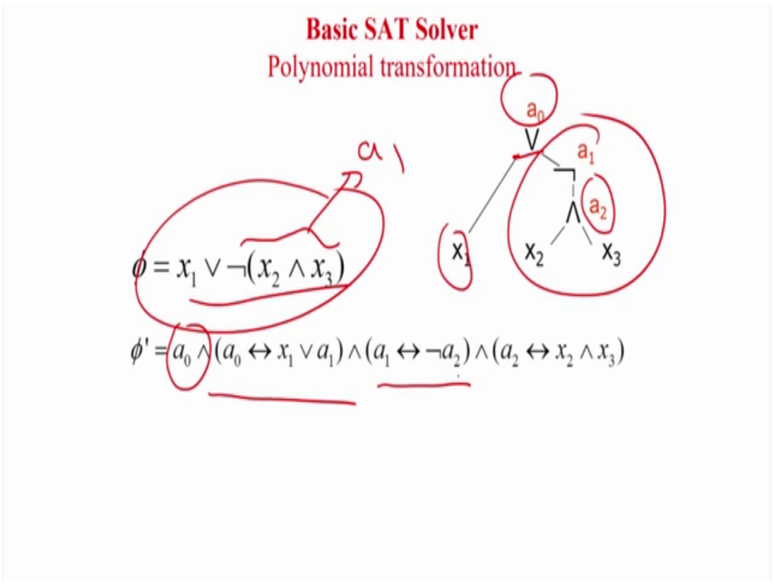
.

But again as I told you sat is the MP complete problem. So, I mean exhausting programs will give you the true solutions, but if the very exponential problem very difficult to solve. So, people have heuristics and we are not going to study heuristics just the pointers, but basically you should understand that given a CNF or a conjunctive normal form basically how at least a simple method to solve it. So, that we can get a idea what is sat and how can you do it?

So, what is the conjunctive normal form we call it conjunction of disjunction therefore, that is we will have ands over here and all the clauses will have some Boolean variables on that variations and there will be or so, the whole things will be satisfiable.

That means each of the clauses has to be true and each of the clauses will be true if anyone of the mutual is true over here. So, that is what is the idea? ok. And then basically what I will show you that how do write it. So, there is a Tseitin way of writing the solutions.

(Refer Slide Time: 58:56)



Basic SAT Solver

Polynomial transformation

$$\phi = x_1 \vee \neg(x_2 \wedge x_3)$$

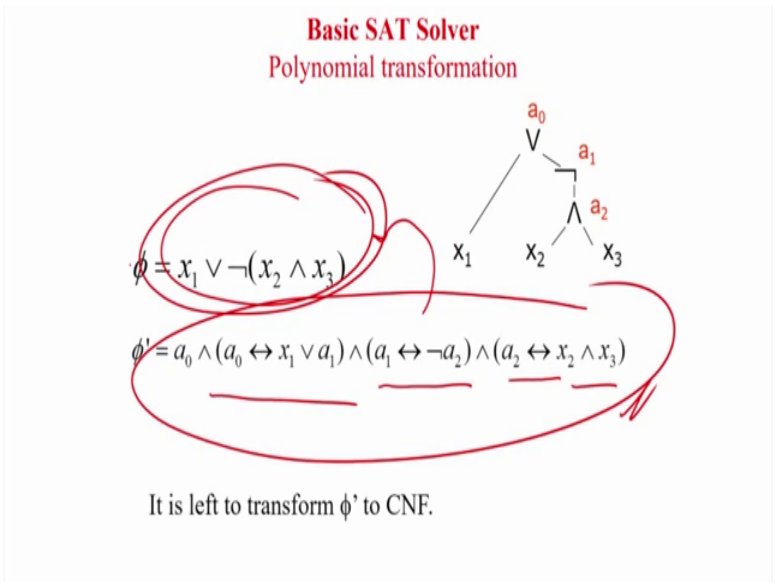
$$\phi' = a_0 \wedge (a_0 \leftrightarrow x_1 \vee a_1) \wedge (a_1 \leftrightarrow \neg a_2) \wedge (a_2 \leftrightarrow x_2 \wedge x_3)$$

So, basically if you have formula like this x or not of x 1 and x 3 how do I write it basically it is something written like this and or a naught if and only if x 1 and or of a 1; that means, this thing is whole represented by a 1. So, I am just writing it in terms of what we have basically written in terms of basically for is bounded model checking the way you are writing this state transitions. So, we are using if and only if clause or if if and only if operator; So, basically, how to write it in terms of a pure Boolean expression that you are just going to reflective.

So, that, because you know that sat can solves these type of things and this is not with if and only this condition, but how this translation can be done, but I am going to give you gist. So, I am writing this. So, this 1 is the case now again this will be a and. So, now, what I am doing. So, I am saying and is equal to not of a 2; So, and if and only if not of a 2 and then what is a 2 and a 2 is equivalent to x 2 and x 3.

So, this is way these how I have written this Boolean formula in this terms rather we are showing how they can be written rather in case of bounded model checking we write in this fashion and then we have to translate it to this fashion. So, that we can go for simple sat solving which basically works on first type of formulas, but we have shown that if this is satisfiable then only this is satisfiable and vice versa that is this is that is phi satisfiable if and only if phi dash is satisfiable that (Refer Time: 60:28) basically equivalent in that terms.

(Refer Slide Time: 60:31)



Basic SAT Solver

Polynomial transformation

$$\phi = x_1 \vee \neg(x_2 \wedge x_3)$$

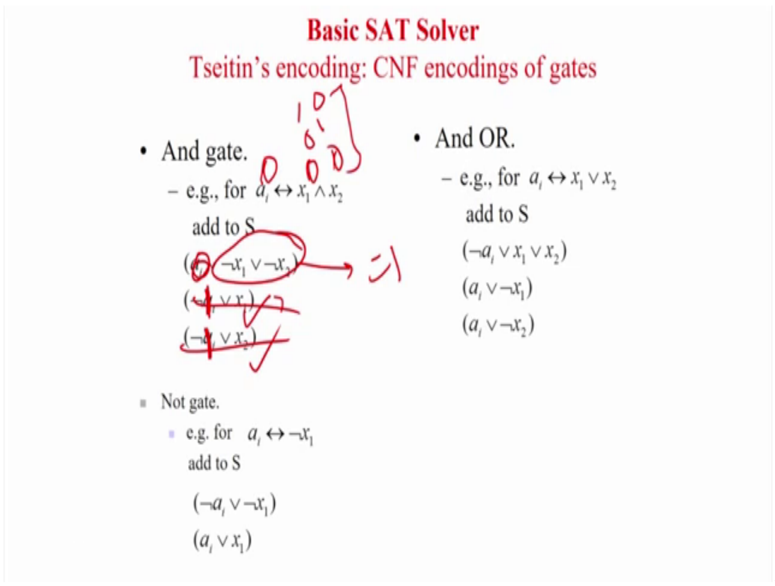
$$\phi' = a_0 \wedge (a_0 \leftrightarrow x_1 \vee a_1) \wedge (a_1 \leftrightarrow \neg a_2) \wedge (a_2 \leftrightarrow x_2 \wedge x_3)$$

It is left to transform ϕ' to CNF.

a_0
$\vee$
a_1
$\neg$
$\wedge$
a_2
x_1
x_2
x_3

So, basically so, what I told you that basically that is why this has to be phi prime has to be transform in to CNF. So, it is a CNF. So, this can be solved by a satisfiable dissolver. So, if you have this is how we write about this transition relations in if in avoid a basically you are bounded model checking you need to transform this and our job will be done.

(Refer Slide Time: 60:48)



Basic SAT Solver

Tseitin's encoding: CNF encodings of gates

- And gate.
  - e.g., for $a_i \leftrightarrow x_1 \wedge x_2$
    add to S
    $(a_i \vee \neg x_1 \vee \neg x_2)$
    $(\neg a_i \vee x_1)$
    $(\neg a_i \vee x_2)$
- And OR.
  - e.g., for $a_i \leftrightarrow x_1 \vee x_2$
    add to S
    $(\neg a_i \vee x_1 \vee x_2)$
    $(a_i \vee \neg x_1)$
    $(a_i \vee \neg x_2)$
- Not gate.
  - e.g. for $a_i \leftrightarrow \neg x_1$
    add to S
    $(\neg a_i \vee \neg x_1)$
    $(a_i \vee x_1)$

So, I will just give you idea for and gate and then you can easily extrapolate for others. So, this is how we write ai equal to x 1 or and x 2 done.

Now what how it is going to be converted into a Boolean CNF formula I write it as this way ai or not of x 1 or not of x 2 not of ai and or xi not of ai or x 2. Now you will see how it holds basically let us assume that a 1 equal to 1 a 1 is ai is equal to 1 then; obviously, this has to be 1 and 1.

So, if I make a 1 equal to 1 then my virtue of this I did not think about it this clause is true, but when a ai is 1 this guy is equal to true and this guy is equal to false sorry the both of them are 0 0 false then to make this clause true sorry to make this clause true and true you should have the value of 1 and 1 exactly.

Otherwise because these are all ampersand then this will be false. So, where directly you can see that I have to have x 1 equal to 1 and x 2 equal to 1. Similarly let us try to make ai equal to 0. So, in this case it can be 0 0 0 1 or 1 0.

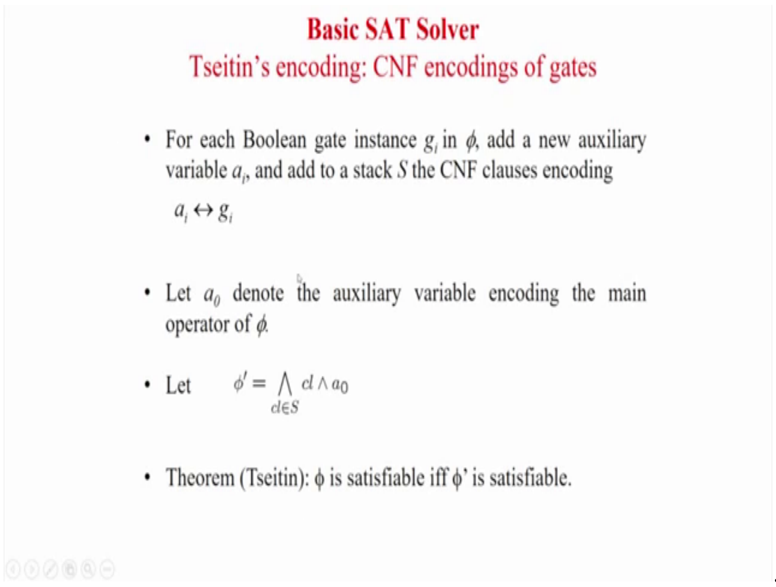
So, immediately if I make ai equal to 0 this guy is becoming 1 and this guy is becoming this guy will becoming 1. So, immediately this and this are true fine. So, I need not think about it then ai is becoming false then only these 2 has to be turn to 1 1. So, now, this has to this 2 taken together has to give the answer is a 1 to make the satisfable.

So, how can not of ai xi or not of x 2 is true be made true there will be true always unless and until both of them are 1 1. So, basically this one is now (Refer Time: 62:27). So, I can easily translate this and gate relationship if and only if way of writing to this then basically this is CNF.

So, all the terms you have written in this case if you look at it in terms of this one in terms of this relationship in caries an xor, but for all gates we can have a similar formulation. So, that we can the whole thing can be transferred in to a CNF formula and then you can sat can solve it for you. So, again I am not going to discuss in details. So, this is again if a or gate and then this is basically for your and gate.

So, similarly you can easily find out that; that means, you have to write that is called ampersand which are not written. So, in that case for this one will be we get in terms this one will be getting in terms and it is (Refer Time: 63:07).

(Refer Slide Time: 63:08)



Basic SAT Solver
Tseitin's encoding: CNF encodings of gates

- For each Boolean gate instance g_i in ϕ, add a new auxiliary variable a_i, and add to a stack S the CNF clauses encoding $a_i \leftrightarrow g_i$
- Let a_0 denote the auxiliary variable encoding the main operator of ϕ.
- Let $\phi' = \bigwedge_{cl \in S} cl \wedge a_0$
- Theorem (Tseitin): ϕ is satisfiable iff ϕ' is satisfiable.

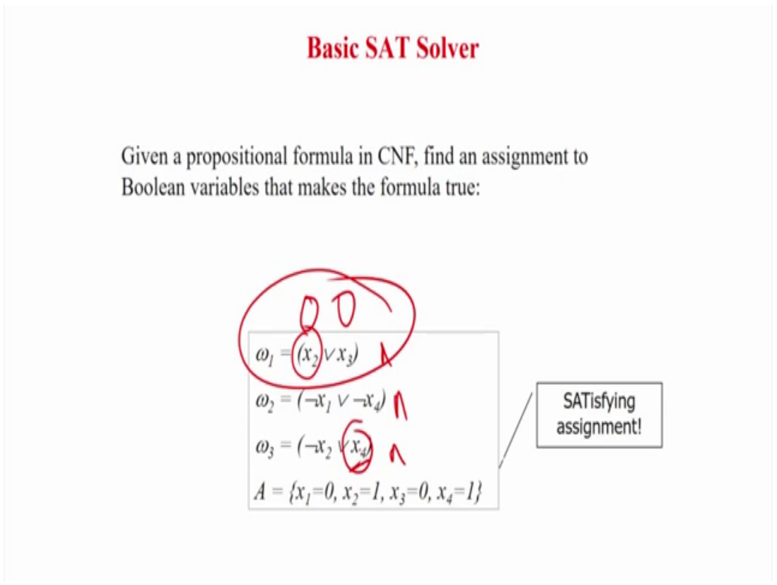
.

So, now that is what is being saying that it each Boolean gate, basically which you are writing in terms of this equivalent formula, we can write it in terms of some CNF Boolean normalized form and then you can have an ampersand of and all these and then the whole formula, which are lot of this if and only if for if and only if basically operator will be converted into a Boolean function or the CNF. And then already we know this theorem we can refer it. So, this equivalent means if phi is satisfiable if and if not if and only phi plan is satisfiable.

So, basically the idea is that you have to write even if you write the relationship of the Boolean model checking boundary model checking even you having this clauses, still we can easily translate them into the CNF form using this structures and then finally, we will give this 3 CNF form to be a statisfiability solver.

And solver will tell you if it is satisfiable what is the values it satisfies it and of course, I have to tell you that there can be multiple values of the Boolean variables which can satisfy it it will report you and those things you can use for to check the counter examples.

(Refer Slide Time: 64:10)



Basic SAT Solver

Given a propositional formula in CNF, find an assignment to Boolean variables that makes the formula true:

$\omega_1 = (x_2 \vee x_3)$

$\omega_2 = (\neg x_1 \vee \neg x_4)$

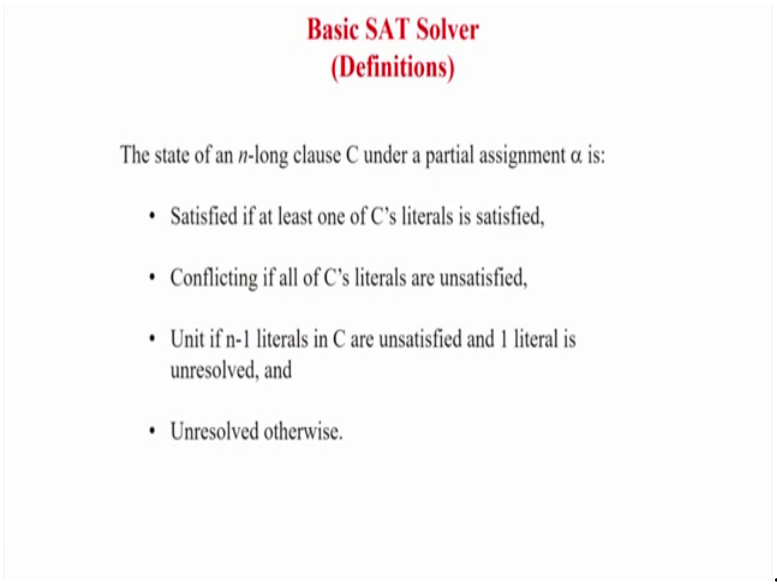
$\omega_3 = (\neg x_2 \vee x_4)$

$A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$

SATisfying assignment!

But if it is not satisfiable that also it will tell that is not satisfiable then basically you will find out that the formula being verify is 2 and there is no counter example, just a simple example. So, basically if you see as I as you discussing this is the CNF. So, these are the some literals and these are the clauses and this is the satisfying assignment.

(Refer Slide Time: 64:25)



Basic SAT Solver (Definitions)

The state of an *n*-long clause C under a partial assignment α is:

- Satisfied if at least one of C's literals is satisfied,
- Conflicting if all of C's literals are unsatisfied,
- Unit if n-1 literals in C are unsatisfied and 1 literal is unresolved, and
- Unresolved otherwise.

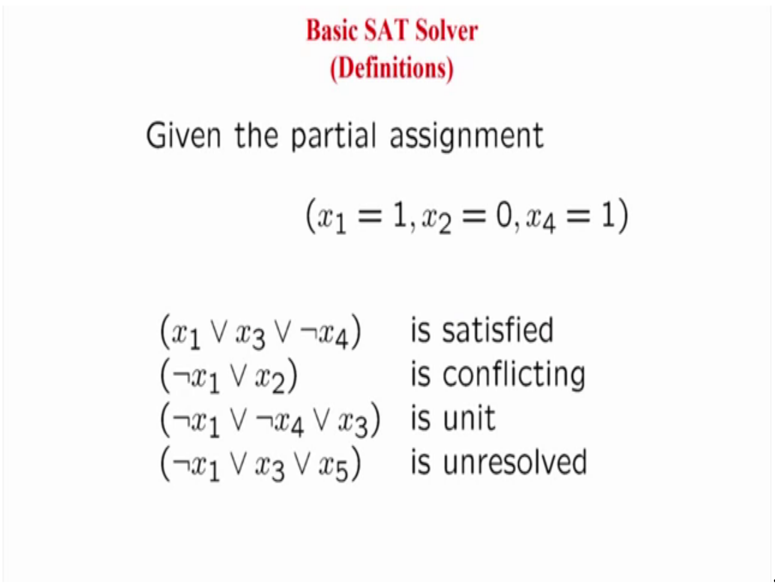
So, there are some standard definitions as I told you satisfy the if at least 1 of the C's literals is satisfied because is in this block any one of the literal has to be satisfied conflicting. So, conflicting if all of these C's literals are unsatisfied.

That means if this is 0 then this 0 then both of them are conflicting then both of them are unsatisfied. So, this clause becomes a conflicting clause and actually this one will make the Boolean that formula unsatisfiable, because they are all ampersands. So, it will be an un this will be basically your non satisfiable it will become a unsat.

So, if you have a conflicting this then this is a unsatisfiable then everything will be unsat problem, because conflicting is all the literals are unsatisfiable it is 1 way that is called as unit, if unit means at least 1 literal you have not assigned any one of the literal is not yet to be assigned and all others are conflicting; that means, this 1 is basically unit means it has to be true to make the clause 2.

So, that the overall is satisfieable. So, unit means the last resolve and the unresolved means some of the Booleans variables are not got their values.

(Refer Slide Time: 65:23)



Basic SAT Solver (Definitions)

Given the partial assignment

$$(x_1 = 1, x_2 = 0, x_4 = 1)$$

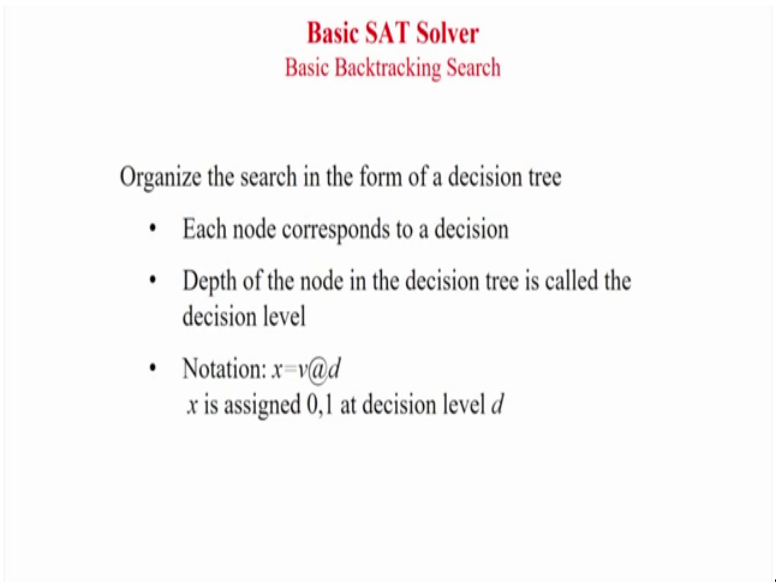
$(x_1 \vee x_3 \vee \neg x_4)$	is satisfied
$(\neg x_1 \vee x_2)$	is conflicting
$(\neg x_1 \vee \neg x_4 \vee x_3)$	is unit
$(\neg x_1 \vee x_3 \vee x_5)$	is unresolved

Very simple example x equal to 1 x equal x 1 equal to 1 x 2 equal to 0 and x 4 equal to 1. This satisfied why because x 1 is 1 satisfies the whole clause, conflicting because x naught is equal to 0 sorry x 2 is equal to 0 means this is equal to 0 and x 1 equal to 1. So, this is 0. So, basically this is conflicting because none of them is true. So, if none of them is true means this is 1 as all the process basically as ended.

So, this will make if an unsat a non satisfiable. So, it is conflicting unit means what you see this S S 3 value we are not clear of, but all others are false this is 0 based on this x 4 is 1.

So, this is also 0. So, the truth or the satisfiabsility of this whole Boolean that for formula depends on the 2 topics, but this is unassigned. So, basically this is called a unit; that means, to make the formula satisfiable x 3 has to be 1 in this case. And unresolved means there are some variables x 3 and x 5 this done is a not being given. So, it is basically some definitions, which you need to know when we will be looking at this sat solver.

(Refer Slide Time: 66:23)



Basic SAT Solver

Basic Backtracking Search

Organize the search in the form of a decision tree

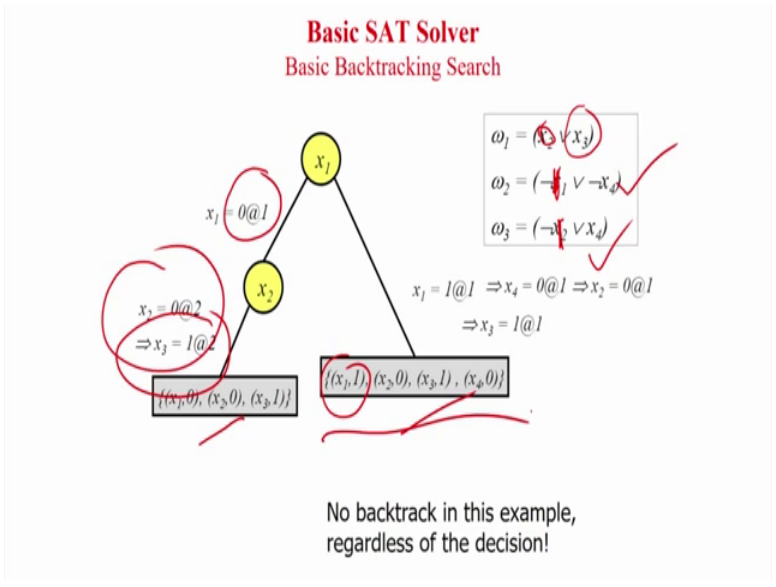
- Each node corresponds to a decision
- Depth of the node in the decision tree is called the decision level
- Notation: $x=v@d$
  x is assigned 0,1 at decision level d

.

Now, what I am going to do, now I am not going to going to any heuristics rather I am going to give you the most worst algorithm that is explicit enumeration and we are going to see whether the Boolean formula is satisfiable or not.

So, in this case what we make a decision tree and each of the decision 3 we go for a depth 1 2 3 4 and each of the depth we try to basically if the variable some values like 0 and 1 and let us see whether it is satisfiable or not. This is the exhaustible algorithm nobody will use it will blow up the complicity immediately, but still it will you give you a picture that how sat is being solved basically.

(Refer Slide Time: 66:54)


Basic SAT Solver
Basic Backtracking Search
$\omega_1 = (x_2 \vee x_3)$
$\omega_2 = (\neg x_1 \vee \neg x_4)$
$\omega_3 = (\neg x_2 \vee x_4)$
x_1
$x_1 = 0@1$
x_2
$x_2 = 0@2$
$\Rightarrow x_3 = 1@2$
$\{(x_1,0), (x_2,0), (x_3,1)\}$
$x_1 = 1@1 \Rightarrow x_4 = 0@1 \Rightarrow x_2 = 0@1$
$\Rightarrow x_3 = 1@1$
$\{(x_1,1), (x_2,0), (x_3,1), (x_4,0)\}$
No backtrack in this example, regardless of the decision!

So, we are taking this 3 clauses and the values. So, at x 1 we are starting. So, we are putting x 1 equal to 0 at the level 1 0 at the rate of 1 means basically we are looking x 1 equal to one. So, if you make x 1 equal to 1 then basically this is going to be a0. So, I am making x 1 is equal to 0 at the level 1. So, if x 1 equal to 0 if x 1 is equal to0. So, basically in this case this is going to be a 1 and. So, this is satisfied you need not bother about at all and lower else there is x 1. So, I am very good.

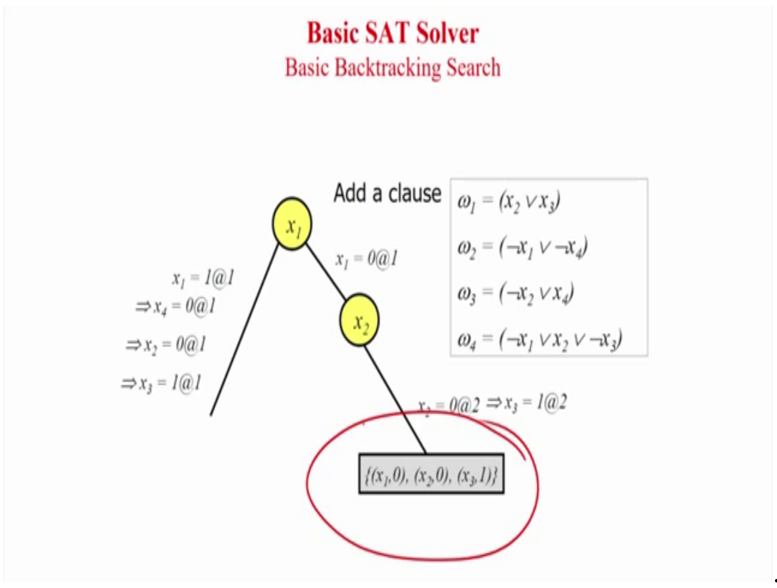
So, I am going to the next level. So, next level I am making is a arbitrary choice x 2 equal to 0 if x 2 equal to 0 then I am putting it is a 0 and this one is going to be a 1. So, this 1 is satisfied now you can see that x 3 as become a unit. So, you have to make it absolutely equal to 1 to get the answer to be satisfied.

So, that is whole thing is satisfied. So, immediately we will write x 3 is equal to 1 on this level and your job is done basically. So, x 0 equal x 1 equal to 0 x 2 equal 0 and x 1 equal to 1 is a satisfiable operation, but this I am telling you it is a very bad algorithm and a worst case complex will be always exponential because you are trying to explore everything and we are starting from the sequence x 1 x 2 x 3.

There can be other sequences like x 2 x 3 x 2 x 1 all combinations all possibilities of permutation combination has to be tried to find a satisfiable. That is why people use heuristics, but this I am just going to show you to know that what is satisfiability and what is the other algorithms to do it. Similarly we can also start with a 1 equal to 1 and then again you can get another satisfiable values, why as I told you that there can be multiple ways of Boolean function can be satisfiable this is the 1 set and basically this is another set.

So, here people started with x 1 equal to 1 and then you can repeat it you will find out the values, but life is always not very simple you can always have some kind of clashes it is you need to be backtrack.

(Refer Slide Time: 68:34)



Basic SAT Solver

Basic Backtracking Search

Add a clause

$\omega_1 = (x_2 \vee x_3)$

$\omega_2 = (\neg x_1 \vee \neg x_4)$

$\omega_3 = (\neg x_2 \vee x_4)$

$\omega_4 = (\neg x_1 \vee x_2 \vee \neg x_3)$

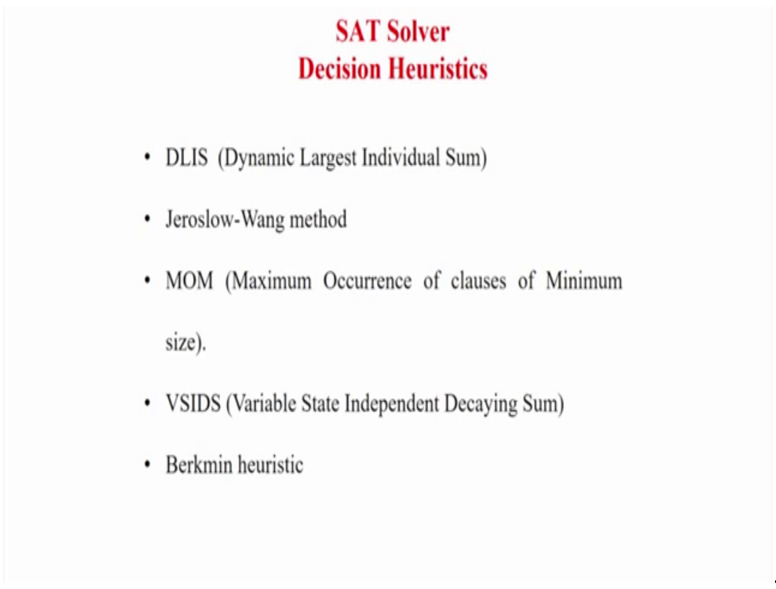
These are very obvious I think you can also appreciate and if you take such a I mean worse I mean such a means rudimentary kind of algorithms of course, you will have clashes and again you have to re track and of course, this can make it thinks exponential.

So, I am taking a new example with some added clause. So, I will go with x 1 equal to 1. So, if x 1 equal to 1 at level 1. So, this is going to be 0. So, immediately you have to make x 4 equal to 0. So, this will becoming 1 then anywhere x 1 is there. So, x 1 is equal to 1. So, this will be immediately 0 then we are x 4 equal to 1.

So, not an issue fine right. So, now, x 4 equal to 1 we have done; So, anywhere x 4 no so, x 4 is equal to 0. So, this is 0. So, immediately x this one is 0. So, immediately this 1 is x 2 is becoming an unit. So, immediately x 2 is equal to 0 right. So, x 2 is equal to 0 otherwise this 1 will become false. So, if x 2 is equal to 0 then immediately this is going to be 0 now x 3 is going to be an unit. So, if x 3 is to be a unit then x 1 has to be of course, we have saying that x 2 is equal to 0. So, x 2 is 0 means this 1 is going to be a 0. So, now, we have 2 units x 3 and not of x 3. So, that 2 are units which are conflicting.

So, you are going to get a conflict over here. So, these are conflict basically. So, this is the conflict because this 1 will tell x 3 is equal to 1 and if you make a x 3 is equal to 0. So, the clause 4 will be unsatisfiable there is a conflict and so, you have to conflict and you have to start the other direction. So, other direction you can start with x 1 equal to 0 and I am not going to illustrate again. So, then again you are going to have something and then it is a basically satisfiable operation.

(Refer Slide Time: 70:15)



SAT Solver
Decision Heuristics

- DLIS (Dynamic Largest Individual Sum)
- Jeroslow-Wang method
- MOM (Maximum Occurrence of clauses of Minimum size).
- VSIDS (Variable State Independent Decaying Sum)
- Berkmin heuristic

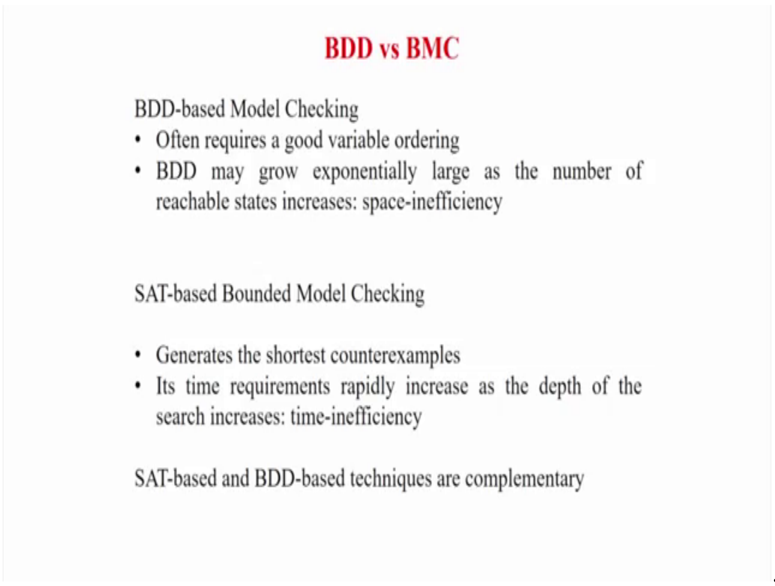
So, basically if you doing this manner it is going to lead to a very big tree and everything is going to take an exponential amount of time in the over space and this is not going to solve anything, but this is given you an idea that if CNF is there then in a tree manner you can find out the satisfiability.

And once it is satisfiable those values will be given, which will act as your counter examples. Now, this is a list lot of heuristics available which basically can solve this sat problem in a much lower complexity, but any way the drawbacks of heuristics will be there, but as I told you generally we for all such MP complete problem the practical way of solving is heuristics.

So, this can solve the sat problem for you in the much quicker fashion and more or less accurate. So, you can have a very optimize solution for model checking of very large systems by bounding the length of search then again there is a lot of questions people ask, whether BDD is good or bounded model checking is good there are lot of ways some people say that bounded model checking of BDDS of course, very is accurate always correct and complete.

But again it says that we need to have a very good variable mod ordering, otherwise it may explode because the size of the BDD is always depends on a good variable ordering. And still people say that as you are still enumerating everything people cannot go beyond state of to the power 1 20 as I have shown you, but for a real case things are much much bigger, but in bounded model checking it is says that it generate shortest counter examples.

(Refer Slide Time: 71:35)



BDD vs BMC

BDD-based Model Checking
- Often requires a good variable ordering
- BDD may grow exponentially large as the number of reachable states increases: space-inefficiency

SAT-based Bounded Model Checking

- Generates the shortest counterexamples
- Its time requirements rapidly increase as the depth of the search increases: time-inefficiency

SAT-based and BDD-based techniques are complementary

Because we are interesting to find out for the first place where the error is and generally errors occur quickly. So, why need to search the entire state space. So, go for a bound and solve lager state space. So, both have their pros and cons. So, basically we tell that basically they are actually complementary approaches. So, both of them has their own merits and demerits.

But, best thing we can get if you can have some kind of high level BDDS and we can do model checking on that at a very abstract level in the BDD bounded level model checking as you have seen is also the bit level. So, can you move to the abstraction level and try to solve all this problem. So, these are research angle which I feel all of you should pursue as if you mainly feeling in this course basically this is also the last lecture of this course basically in this course you have try to discuss about the card algorithms, which is traditional and then you have seen what are the different optimization problems and how we are changed or modified this traditional algorithms so, this complexity can be handled.

But still for many of the cases we have not reached desired level like if you take verification, you have not even reached level where you can verify the whole soc and noc in a p in a group. Basically because either you have to go by symbolic model checking or if you are going for the abstract diagrams with what a very good algorithms of model checking in those high level abstracted diagrams so, there are lot of gaps.

Similarly, in the way of testing, similarly in the way of designing, and all these things we have found out gaps and there is a lot of entire course in itself is just gives you some directions in which you can pursue your research when you are doing for card algorithms. So, traditional card is very well established the optimization formulas I have told you in this course and also I have told how we are attacking them.

And also the gaps that remains have also been identified in this course. Now if an we are actually lot also uploading we will be uploading lot of references papers and direction. So, the goal of this course one of the more important goal of this course is that you please pursue or try to take the important problems, which are the which have to be solved in this optimization domain, because when you are still when you have going for very large scale multi processors nocs socs and this is your this size will be becoming larger and larger. And our algorithms are not actually apart to meet them.

So, the lot of optimizations important problems remains to be solve. So, in this course we have just given pointers to them and we request you that you take interest in this and try to pursue your research in that.

So, again thank you all for attending this courses and we will hope that you will pursue research in optimization of channel card for digital VLSI design.

Thank you from our side.