Foundation of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

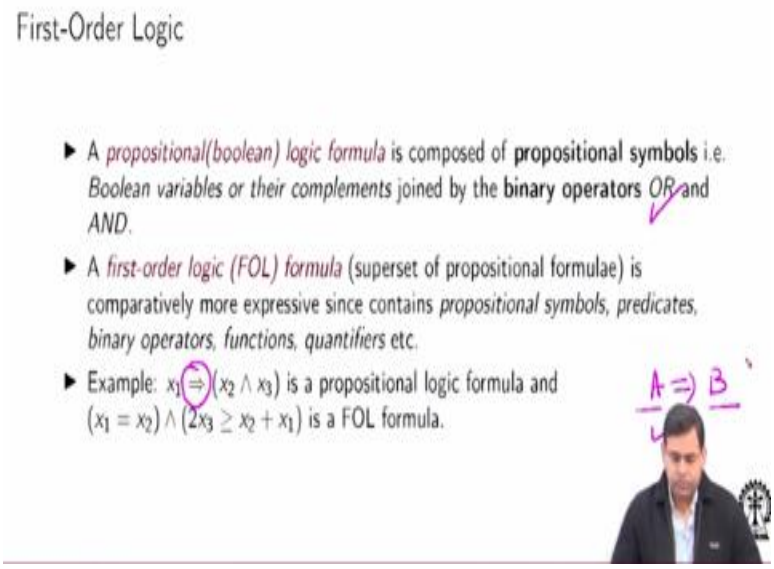
Indian Institute of Technology, Kharagpur

Lecture - 36

Reachability Analysis (Continued)

Hello and welcome back to this lecture series on foundations of cyber physical systems. So, I believe in the last lecture we showed you some examples of timed automata-based modelling of simple systems like train gate control and elevator control systems like that. So, today we will be going back to this topic of reachability and how reachability of hybrid automata can be checked in practice using some specific kind of solvers like satisfiability modulo theory or SMT solvers.

(Refer Slide Time: 00:56)



So, just a small recap on these concepts like what we are trying to do here is we will try to write a formula, a logic formula to be precise which captures the evolution of a given hybrid system model. So, we will be borrowing elements from Boolean logic and first order logic. So, in general a Boolean logic a propositional logic formula is composed of proposition of symbols or Boolean variables and their components that join which are kind of joined by the binary operators or, and, also you will have the negation operator.

So, just as an example you may be having certain variables which represent states of the automaton let us say I have two states and there are corresponding variables so if q 0 is true then I am in the state q 0 that is how it will mean. And separate individual facts of the automata will be defined accordingly with suitable propositions and multiple facts holding together to be true will be similarly defined by their conjunction.

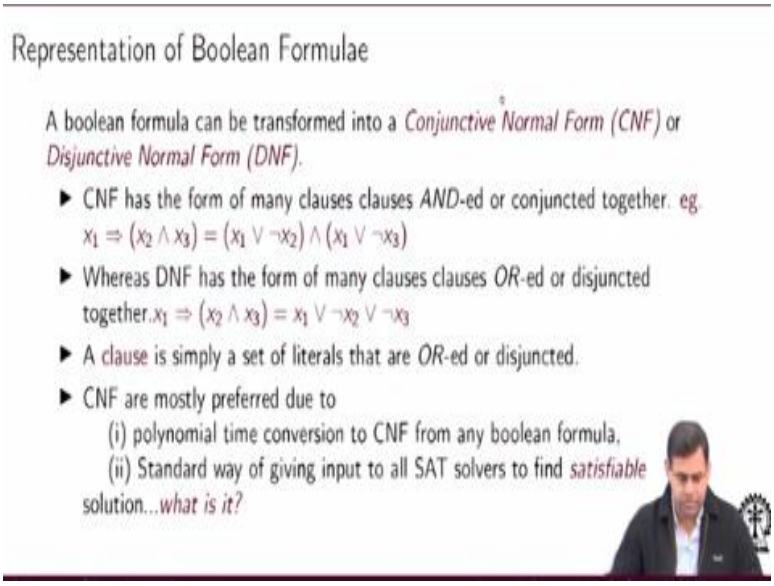
Or any one of them to be true will be kind of represented by their disjunction OR and AND something to be false will be a negation of that propositional proposition or a Boolean predicate. And all these things will combine to give you a propositional logic formula. And we will also use first order logic essentially what it does is it brings in propositional symbols predicates binary operators also functions and quantifiers.

We are not going into the detail of how to formally define first order logic formula but we will actually show in this context how those artifacts are used in practice. Let us say for example. So, this is another Boolean connective which is logical implication that means I mean if x 1 is true the right hand side must be true. So, you can actually draw a truth table of an A implies B. So, these are logical implication.

So, that would mean that whenever A is false let us say you write it in the general form A implies B both sides can be propositional formula. And what this means is that whenever A is false this is vacuously true the enter statement is true and whenever e is true then B needs to be true for the entire statement to be true. So, that is an example and you can have similar examples of first order logic formula and they can also have quantifications over certain variables.

So, I mean you can have quantifications like well there exists some value some variable x_1 for which the following holds something like that and also another important thing that you can see here is we are using linear relations here. For example, when I am connecting like this, I am using a linear relation here that also means that well here I am not interpreting this variables x_1, x_2, x_3 as Boolean variables but rather they are kind of real variables or integer variables depending on which theory we are going to use whether the theory of integer arithmetic or real arithmetic. And we are use we are combining them to generate linear relations here using or operating using binary operators which are primarily of this set. So, as you can see this is where we extend from pure propositional logic to other kinds of logics where these in first order logic we are allowing these complex predicates build as linear relations over integers or reals. And that is precisely what we will be using in our SMT formulations.

(Refer Slide Time: 05:02)

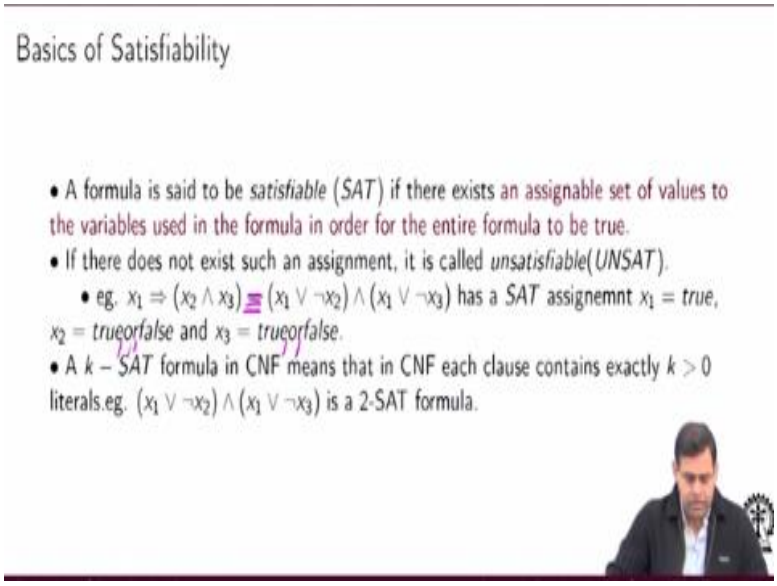


So, a Boolean formula can be transformed into either a conjunctive normal from CNF or a disjunctive normal from DNF. The CNF will have the form that, there are many clauses which are AND-ed or conjuncted together. So, here we have some examples so I can have something like this or when you are talking about disjunctive normal form, we can have many clauses which are odd together or they are disjuncted together.

So, you have some examples here. So, by the way what is the clause? The clause is simply a set of literals that are kind of odd or disjuncted in the DNF form whereas in CNF you will have them as kind of AND-ed. So, typically for SMT solving I mean checking satisfiability of this kind of formula, by the way satisfiability means finding out a value assignment to these variables for which the entire logical formula evaluates to true.

So, for checking such satisfiability we will be using different kinds of solvers which are quite well known and available and these solvers typically will take the logical formula as input using a CNF representation.

(Refer Slide Time: 07:02)

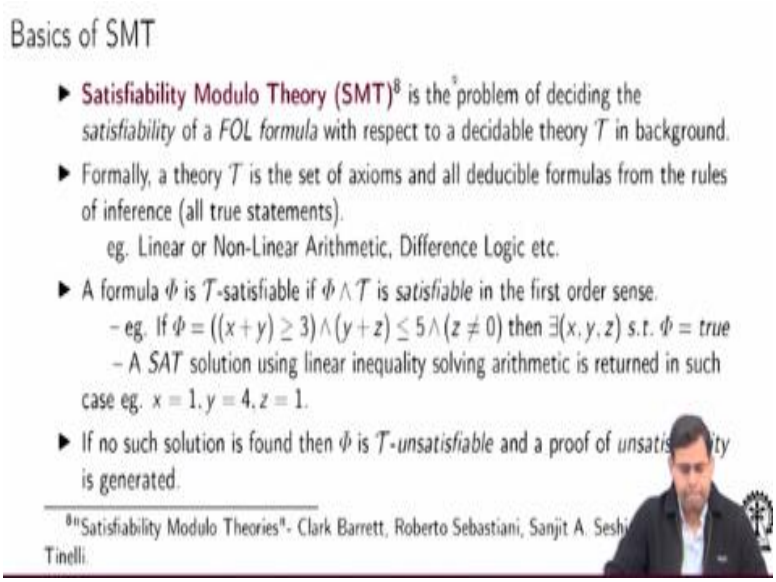


So, we will say that a formula is satisfiable like I was saying if there exists an assignable set of values to the variables in the formula in order for the entire formula to be true. And if such an assignment does not exist then we will call it an unSAT or an unsatisfiable. So, for example let us say you have x_1 implying x_2 and x_3 you can also write it in this form I mean here typically we will be using the equivalence symbol this is this formula is equivalent to this one.

And this is a SAT assignment that well if I put x_1 = true and whether I mean x_2 is true or false or x_3 is true or false does not matter, the entire formula has to be true. Now you can have specific

kind of forms for example I can say that a stat formula in CNF form is basically a k SAT if in each of these CNF clauses you have exactly k literal. So, for example these are two SAT formula that means in each clause we are allowing two literals. A literal is basically Boolean variable or its negation.

(Refer Slide Time: 08:19)



Now if you remember here we showed some formula like this so you are bringing in I mean not simple I mean propositional variables but we are also bringing in other variables in the form of linear inequalities. So, let us understand let us go deeper what that is all about. So, what we do is we use a special class of solvers called SMT solvers or satisfiability modulo theory solvers. And they solve this problem of SMT which is to decide the satisfiability of a first order logic formula with respect to some decidable theory in the background.

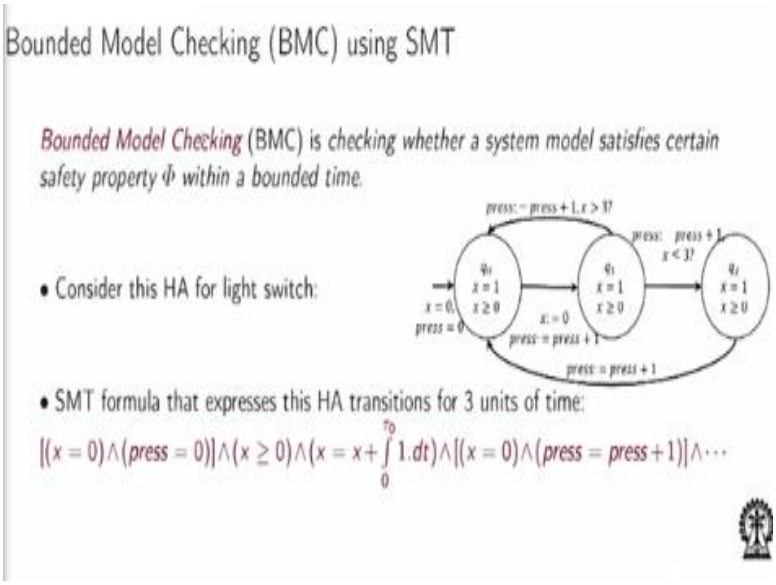
So, what do I mean by decidable theory? For example, you have theory of real closed fields, theory of linear arithmetic. These are theories in which there exists proof techniques through using which a given a formula belonging to this theory the proof technique can tell you through its own algorithm or decision procedure whether that formula is true or false. So, this SMT solvers will use in its back end many such theories available and it they will invoke those theories to check whether a given SMT formula is true or false. So, this is not a comprehensive course on SMT solving. We are trying to show that how that can be just used in practice to solve hybrid automata

problems. That is all we are doing in a very introductory manner. So, suppose you have a theory I mean which is which is given like the theory of linear arithmetic, difference logic etcetera.

And for a given hybrid automata if you are so able to create a corresponding logical representation of all possible runs of the automata using one of the theories. Then if you can put a corresponding query to a solver, you will expect the solver to answer that query. So, a formula phi will be satisfiable with respect to some theory in the first order sense. So let us say you have this formula like this so, you have x and y, two variables let us say these are integer variables.

And you have this constraint ANDed with this constraint and another constraint then the question is does there exist value assignment of x, y and z integer. So, that this is true so this is the candidate SMT formula which can be solved by when and when the SMT solver invokes the theory of a linear arithmetic to check if this formula holds. So, that is the solution if, there can be a satisfiable solution to this set of linear inequalities and that the solver can return it like this. So, the question is in case the solution is not found then this formula is unsatisfiable and the proof of unsatisfiability may be generated by the solver.

(Refer Slide Time: 11:20)



So, what we will do is we will consider bounded number of discrete time steps of evolution of such hybrid automata and try to show that under such bounded number of evolutions or in the discrete in terms of bound in terms of number of discrete transitions taken, whether certain properties are true or false about such automaton. So, let us consider this light switch example. So, you have 3 states.

If you see now, we are considering press as an event count so whenever there is a transition, we are increased this press event count by 1. And you have the usual invariance guards and resets that are there in the automata. Now suppose I am trying to model this evolution of the automata for three units of time. So, what I am trying to do is I am trying to write a logical formula that captures all possible evolutions of this automata starting from some initial state which is $x = 0$ and press = 0.

And so as you can see from starting here it can go this way or it can go this way and it can also decide exactly when to take the transitions all these things creates a reach set a reachable set of possible states that means it can be in this state with value of x being this or it can be in that state with value of x being that things like that. So, that is it I want to characterize mathematically can I do it using a SMT formula that is what we are trying to see.

So, like we said it will be nothing but a collection of constraints which will be ANDed together each of these clauses or constraints will be considered will be having a linear inequality typically here if we are using this theory of, I mean linear real arithmetic which should be the case, for this case where we are considering linear hybrid automata. If it is a non-linear hybrid automaton you have to consider solvers. There are solvers which deal such non-linear inequalities also.

I mean they use techniques of approximating such non-linearities with linearities in the back-end stuff like that, but the solvers there are solvers which can also handle non-linearities. So, let us see

if I can write constraints and clauses which will capture this evolution. So, what we are showing here is we start from this initial state x = 0 and press = 0. And we must go to this state and satisfy this constraint that x greater than equal to 0.

And then x is allowed to evolve up to some non-deterministic time τ_0. So, you write that flow equation here in terms of this integral. So x will keep on getting updated following this rule and now you see when I come here x must be reset. So, when x gets resets, we will have AND x = 0. Now that looks kind of if you see here, we had x = 0 then x greater than equal to 0 then x is getting updated and here x = 0.

And then you are having a press = press + 1 because this variable is going to change its value from 0 to 1. So, we are trying to capture all those issues here. Now the question is if you see this variable x was getting a positive value due to the evolution and now it is going to be reset. So, this looks pretty inconsistent here. Why because I mean I cannot solve like this because I mean. So whatever I am really missing here.

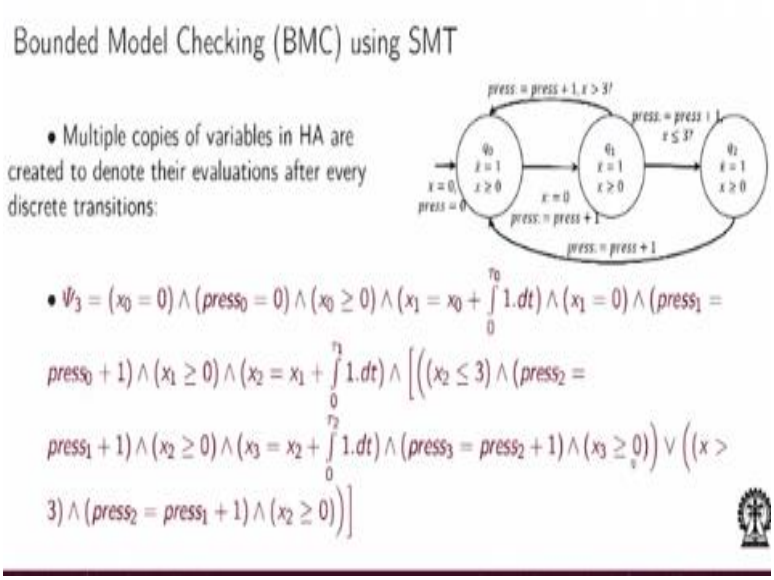
What I am really missing here is well initially x will be 0 then x will be updated and then x is going to be reset. If I am trying to check if I am trying to create a formula where I write like this and conjunct like this, this is clearly not the correct formula which is capturing these entire phenomena because if x is 0 then how can x be nonzero how can x be this one. So, that is very difficult that will actually force τ_0 to be always 0. So, that is very bad.

So, the problem is you see x was initially definitely 0, then x must get positive and then x is reset. So, we have to capture this thing, so the way to do this is we will follow a convention which is single static assignment. That means whenever a write is happening on a variable, we will use that variable in a renamed manner. So, that will be a phenomenon which actually captures that well earlier the variable was there it was getting updated and then some value was forcible it.

So, that so earlier the variable was flowing with time and increasing its value continuously and then at certain point of time that variable was updated through by force which is a right. So, that is not a natural evolution of the variable following the differential dynamics, that is a hard write. That is which is being forced on it. So, we have to make a distinction between these two things to capture them and write them in a logical formula.

So, that is why what we will do is we will introduce renamed versions of the same variable so that, so basically it will introduce more number of variables per single variable here and that will help us give suitable value assignments to different variants of the same variable and accurately characterize this event.

(Refer Slide Time: 16:49)



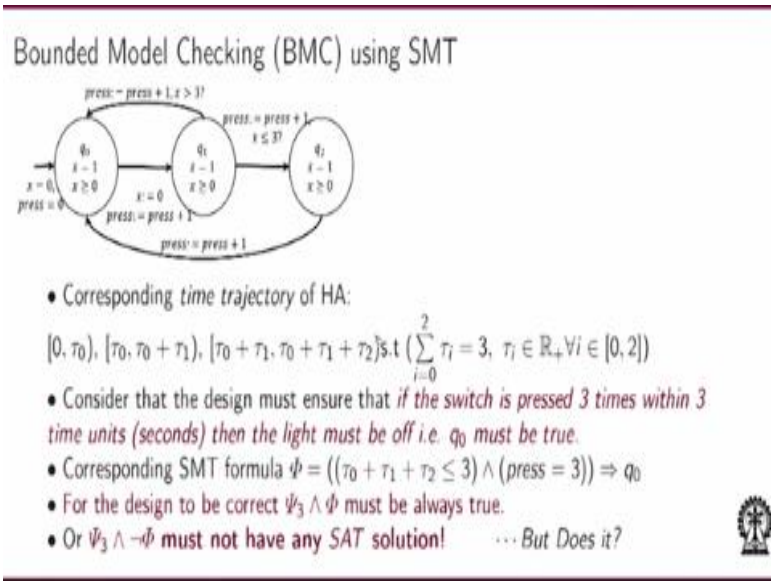
So, what we really need to do is you see whenever a variable is being written we rename the variable. So, here you see x_0 is there so and then we call x as x_0 initially similarly we have press x_0. So, here x_0 is still not written x_0 is a clock constant but when x is getting updated, this update now goes to a renamed version of the variable x_1 this continuous evolution and after that happens x_1 is being assigned to 0 here through the reset.

And then press is again changed so the update goes to our renamed version of the variable to make it press 1. And then here when I am in this state, I will again need to satisfy x_1 greater equals 0 and then x this variable will be again be written there will be a flow dynamics like this here so, you introduce a new variable x_2 and then you update and then you check that the well it must satisfy.

Now from this point it is interesting you see there are two possibilities. It can go this way or it can go that way. So, from here you have to create two different clauses. So, one way would be that well it can go it can satisfy this way and come here. So one clause would be well x_2 less than equals to 3 and then you will have the dynamics of this state getting captured with x_2 greater than 0. The state is evolving and then press is incremented here and all that or you will have the dynamics moving the other way like this which is like well x is greater than 3. If that is satisfied then press is getting into updated like this. And you should have this getting satisfied as x_2 greater than 0. So, here you can see we have an OR which is kind of capturing that well we are going this way or we are going that way. So, that is how more or less is going to work.

So, now you can see that if I have a single initial state, from this single initial state, I can see all the cycles in that automaton that I can traverse and accordingly I can keep on creating this kind of clauses and eventually I should get a formula.

(Refer Slide Time: 19:08)



Now let us get deeper into the system. So, you can have a corresponding time trajectory of this hybrid automata. So, let us say you are spending initially, so if you see here when I created the formula you are also introducing this time variables τ_0, τ_1, τ_2 which are indicating how much time we are spending at which place. So, if you see this formulae, so here well this is fine and then you take this transition that is also fine.

And then you are here you are you are updating press you are satisfying the invariant x_2 greater than 0 and then you are updating your final value and then from here you are again increasing the press event and here when you go you need to again satisfy x_3 greater than 0. Or in the other way it is a direct transition to this state so you do not need to go up to x_3 and only checking up to 2 will suffice it.

So, here you have two transitions in this cycle. In this cycle here you have two transitions so this is a larger clause here and this cycle there is a single transition to an original one. So, that is why you have this clause here. So, I hope we have given you an idea that well how to build this formula what are we doing we are actually doing a lot of things here. We started with a basic Boolean formula like this.

And then we said that well the formulas can be represented either in CNF or DNF form but that is for the solver that is for specific to the solver's input. Then we say that well we can also have first order logic formulas. So, where there are quantifiers functions binary operators etcetera and then for doing, for our purpose specifically for linear hybrid automata what we did is we brought in a formula of a strong the formula in a specific structure, we said that well we have discrete variables and we also have real variables.

And since the linear hybrid automata the real variables will have simple rules like this and they will be updated using this kind of equations this kind of simple ODE's and accordingly their updates will be written using constraints like this and resets and guards will be actual accordingly captured and all possible paths in these automata can be written using this kind of constraints. Because whenever I am taking a path what is happening the corresponding variables.

We will need to satisfy the constraints on the path and those things I can just model using these linear inequalities over the real variables. And finally, we will have this kind of a formula and if I want to check that whether this evolution is possible or not all I need to do is I will query a solver and see the ASCII to give me suitable assignments to these unknown variables. So, that this formula is true.

Then it will say that well this there is a possible set of value assignments for which this evolution of the hybrid automata is possible. Let us dig deeper into this so here if you see the moment, I chose this $\tau_0\tau_1$ and $\tau_3\tau_2\tau_3$ like that I am telling that well exactly how much time I am spending in each of these states symbolically I mean in terms of τ_0, τ_1 and τ_3 like that. So, initially I am in q_0 for 0 to τ_0 interval, then from τ_0 to $\tau_0 + \tau_1$ absolute interval.

Basically tau 1 is the interval and the absolute values of the start and the end of the interval is τ_0 and $\tau_0 + \tau_1$ and similarly in this way I can figure out how much time I am spending in each of these states. Now suppose so now suppose I want this thing to happen that the entire set of transitions should happen inside 3 time units. That would mean the summation of type τ_i should be three.

Now let us say I have give a requirement that the design must ensure that if the switch is pressed three times within three time units then the light must be off, that is q_0 must be true. So, that would mean I am querying this system that does it satisfy this requirement. The requirement is that $\tau_0 + \tau_1 + \tau_2$ less than equal to 3 and press = 3 that means the total number of press events is 3, this implies q_0.

I hope this is clear so this is the formula which is capturing the evolution of the system over three transitions now, because three transitions will give me three consecutive press events. I want to know whether it is possible that if these three press events happen inside 3 time units then the light must be off. Light must be off means I reach here. So this is the corresponding formula which I want to be true.

That whether this is happening so then the question is that well what I am really looking for is whether this formula and this formula they are together true for suitable assignment of values to this x_0, x_1 and other stuff. Whether this is always true sorry and so I am not looking for satisfiability here I want the design to be correct that means I am looking for validity. That means whatever is my way of evolution as long as the evolution is following this structure.

That means I start from the initial state here and whatever is my choice of $\tau_0, \tau_1 and \tau_2$ that means where whether I take the so if I have to have three consecutive events here it is like coming here coming here and coming here. So, let us say I am forcing this path here I am not going into that

path and I am forcing this path here. So, that means I am looking not for this entire formula I am let us say I am looking for this path here.

Because if I am going to this path then that must be x greater than 3. So, even if I have this formula just by my requirement I will not ever be going to this path. So, then that would mean that for as long as my total interval is less than equal to 3 this is fine, this formula is capturing all possible evolutions which is fine. So, what I am really checking is that well inside all these evolutions do I always have the case that with three such transitions I am always going to land up here with three press events happening.

That means whether for all possible such evolutions which satisfy this requirement these things together are going to hold. Now how do I check this? the way to check would be that you check the other thing because this would require you to check for all possible value assignments, so you take the other way. You check whether if there is a way to satisfy ψ_3 and negative ϕ. That means for all possible evolutions of that automata is it possible that there exists some evolution or some specific value assignment.

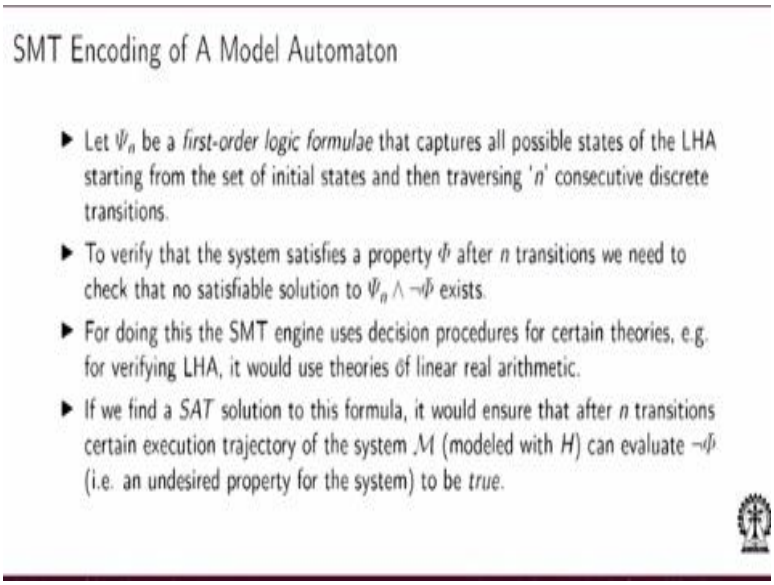
So, that I take the some path in that automata but this formula is not true. So, what I need to say that well this must not have a SAT solution. So, all I need to do is I need to give the SAT solver this kind of formula as input and if the SAT solver can give a sequence of value assignments that means well there is a possible path in that automata. So, that I take three I will press the switch three times within 3 time units and I yet I do not land up in q_0. That is how it works.

So, that is why we have this question here but does it. So, this does not have so this is how the solver trick sheet is going to work that well, if I run it with this if there is such an evolution for which this thing happens that the switch is pressed twice within 3 time units but the light is still not off. Then this will have a solution and otherwise so for a solver you can have these possibilities.

You it will return your value assessment or maybe for solving the formula, the formula may be complex and the solver may be taking way too much time and your memory is a kind of clogged in your system and the solver is do not returning any kind of value assignment. So, then you have a case where it is difficult to say a yes, no answer because it only means that it may be difficult for the solver to find out an assignment but it may not give you an yes to answer.

So, that is like the point when you are stuck. But if the solver gives you a value assignment, then well it is telling you that yes there is a solution to this problem or the solver may find an unsatisfiable proof that means it can terminate saying that well there is no value assignment possible and it can also give you such a proof of that well this is why it is not possible. So, that is what we mention here that the if no such solution is found then it is unsatisfiable and the proof of unsatisfiability is generated. So, that is what we are saying that there may be a yes answer there may be a proof that is generated, the yes answer would give you a counter example that will this is how your requirement may not get satisfied. And that counter example can help you to refine this design and also the thing is that it may happen that the solver does not terminate because of memory or space constant in your system depending on your design. Now just to remember I thought I did not mention, so this is the light switch example we had initially in our lecture here only. So, I hope everybody is able to relate to that.

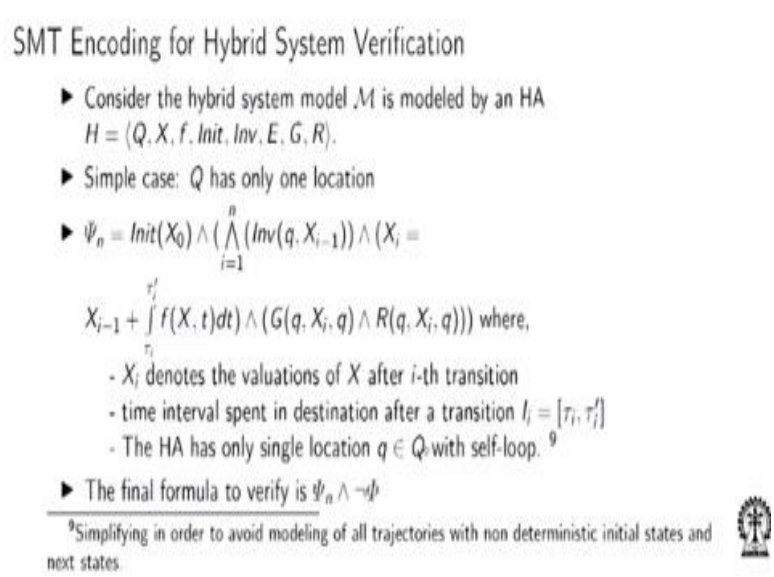
(Refer Slide Time: 29:09)



Now so how do I really encode the SMT of a model automaton like we did here. So, the steps that we followed if we just kind of try to summarize them we are creating a fast order logic formula psi n which is capturing all possible states of the linear hybrid automata starting from a set of initial states and then traversing n consecutive discrete transitions. So, for verifying that a system satisfies a property ϕ always that means that entire thing ψ_n and ϕ is valid after n transition we need to check that well ψ_n and negation of ϕ will there exist a solution to that. So, we are expecting that well if the original thing is valid then there should be no solution and if there is a solution that tells me that well your original hypothesis of this all the evolutions taking n consecutive transition satisfying phi is not true. So, what the SMT engine will do? The SMT solver will use the decision procedures of this for LHA, it will use a theory of linear real arithmetic to deduce the truth or falsity of the formula.

So, if you get a SAT solution to this formula it ensures that after n transitions as there exist certain execution trajectory in this system for which you get to a situation where your requirement is not satisfied.

(Refer Slide Time: 30:32)



So, that is how it will happen now consider this hybrid system. Let us go deeper into this topic. So, and we consider a simple example first let us say. So, now what we are going to see is so what we saw here is well how to model the evolution of a hybrid system and how to use it to check some requirement which is given captured using a formula. So, let us now go deeper into that. First we start with the simple assumption that we are let there be on a single initial location in the automata.

So, let us speak of our so earlier we saw a complex example but now we will do something in a different way we are trying to see how these methods apply to a cyber physical system where you have a plant as well as a controller. So, first start with a simple plant. So let us say you have a single location hybrid automata and you have a transition like this. So, how do I model this? So, the formula which will model all the evolutions here would be the something looking like this.

That you have an initial set of states. So for the initial location all possible initial valuations are captured here using this formula. And then you can have n number of transitions here n number of cell flows being taken. So, in each of these cases you have to satisfy this invariant of the state for the corresponding valuations that are reachable with i number of steps here and then for each of those transitions you need to satisfy this update equation.

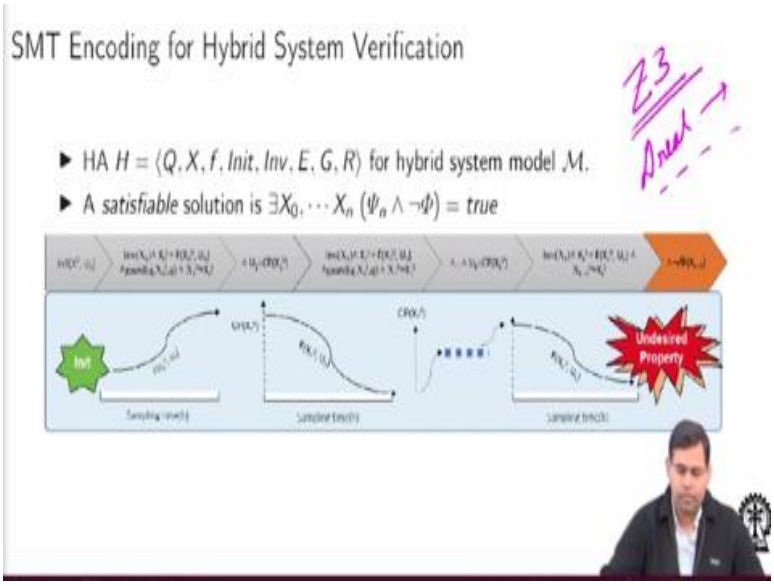
So, as you can see what we are doing is we are introducing new variables with each transition. X i denotes the valuations of the continuous set of variables. So these are set of continuous variables. And X_i is denoting the valuations of X after the ith transition. So, you will need to satisfy this update equation each time. So that is why both these things the invariant and the update equation together are under this AND.

And also, under this AND in each time you will need to satisfy whatever is the guard condition here so you will have an invariant here, you will have a guard here and you will have a reset here so, with every transition the guard needs to be satisfied the reset needs to be satisfied and with each time your valuations will be updated like we saw earlier. So, that was for an example and here we are generalizing it into a generic form.

For simplicity we are assuming there is a single mode hybrid automaton. So, you start from set of initial valuations which are captured by this and every time you take a transition you have to satisfy invariance you have to satisfy and following with those invariance you have to satisfy the flow and with the flow the update will be made with the flow you have an update that is happening to your continuous variables.

And after that these continuous variables must satisfy the guard and also the reset and with this you go to the new iteration for an updated i. So, basically X_i is already computed so now that X_i in the next iteration of i that will be your X_{i-1} and then you go to a modified X_i and like that. So, this is like a general structure of an automata that we show here with a single location and this formula structure is capturing all possible valuations that our automata can get into with n number of transitions here and if I want to say that well this automaton will always satisfy some formula ϕ, I will need to check value assignments to ψ_n and negative ϕ.

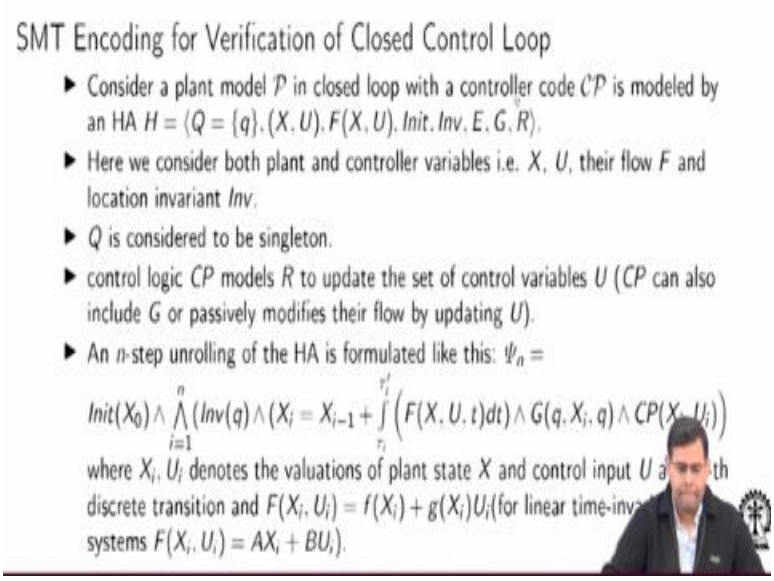
(Refer Slide Time: 34:17)



So, that is how it works so this is a pictorial representation. Like you see that you have a sampling time. So this is your automata and we are trying to figure out exactly these that whether there exists this sequence of X_0, X_1, X_n so that this thing is true and with each evolution your system is changing its trajectory like that following the flow and it may happen that finally it satisfies this undesired property.

If this kind of a path really exists that means there is a possible set of X_0 followed by a possible set of values in X_1 followed by a possible set of values in X_k like that so that phi is satisfied then this path would be given as output by this SMT solver. So, the very popular choice of SMT solver is z3 from Microsoft's website it is free downloadable you can just check this if you are interested. And then for hybrid automata you have D real and there are several other such hybrid automaton. So, this can also handle in non-linear systems.

(Refer Slide Time: 35:24)



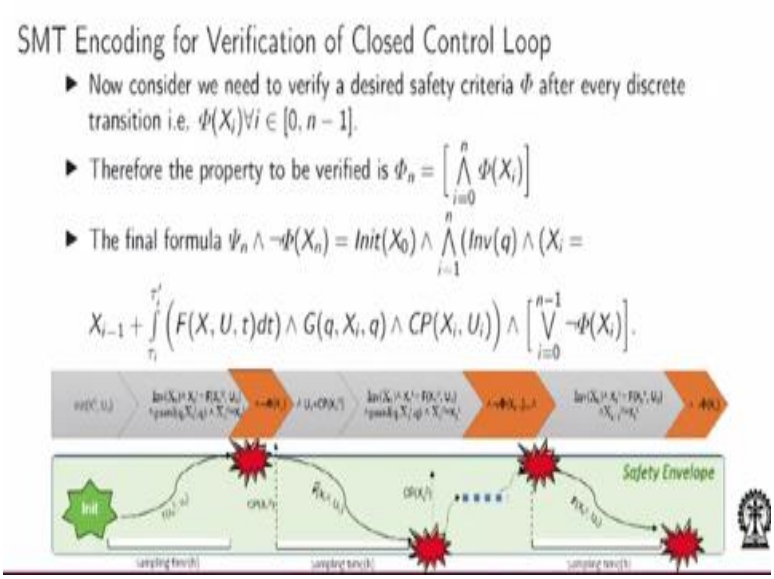
So, suppose now I am talking about a plant along with a controller. So, your controller is a code let it be represented by a function CP. So, here you have your original automaton now what will happen is now in your automaton there will be certain continuous dynamics variables and there will be certain variables let us say in this set U which are going to be updated or written to by the controller program CP.

So, you have plant variables X and U and the flow F and the control variables updates will be specifically for U. Now still we are considering Q to be that singleton set. So, when we are doing that the structure of the formula will be like this the control logic is basically like a reset which is updating only the control variables. I mean typic I can also model it like a guard in case the control logic is just checking things but eventually the control logic is going to modify things also.

Basically, it is a control program it is like a program and it which can be written to a set of clauses which represent updates happening to certain variables and certain conditions getting satisfied. So, all I am saying is given that program you will need to represent it in the form of clauses just like we are representing this hybrid automata into a set of clauses. So, all that will happen is you have the same structure like the earlier one we discussed.

Only thing that we are doing is here you see after the guard, I miss the reset here the reset should also be there the hybrid automata's own resets, and along with that you will have the control program I mean which is kind of here represented as a formula here which will update the variables in U. So, with that you have a formula representation which is showing that well you can have a single mode hybrid automata for a plant and a controller together how a corresponding formula can be made out of that.

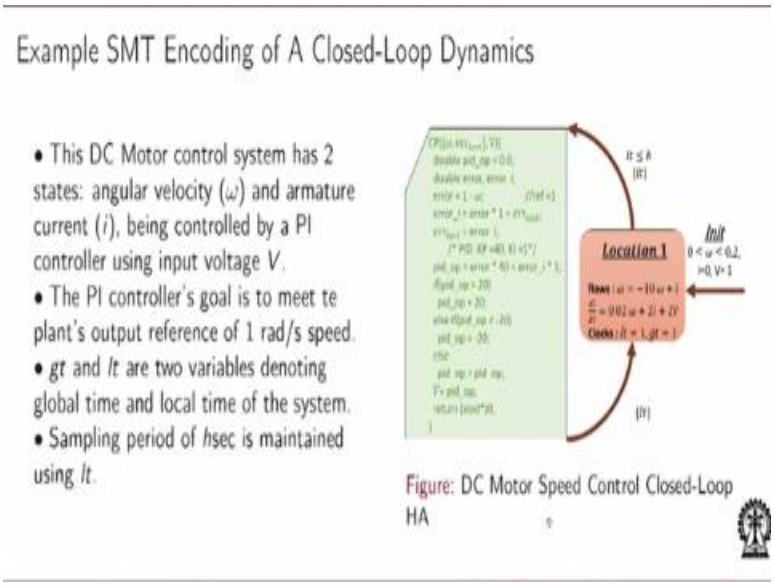
(Refer Slide Time: 37:21)



So, you will again have a formula to be verified which will look like this and the final formula that we that is to be checked for a value assignment we will just look like this here. And maybe you will find that so essentially you have the plant and then the controller again the plant and then the controller together and that evolution. Let us say this is your safety envelope that what this phi is telling you is that while the evolution is happening there is a safety envelope which must be satisfied.

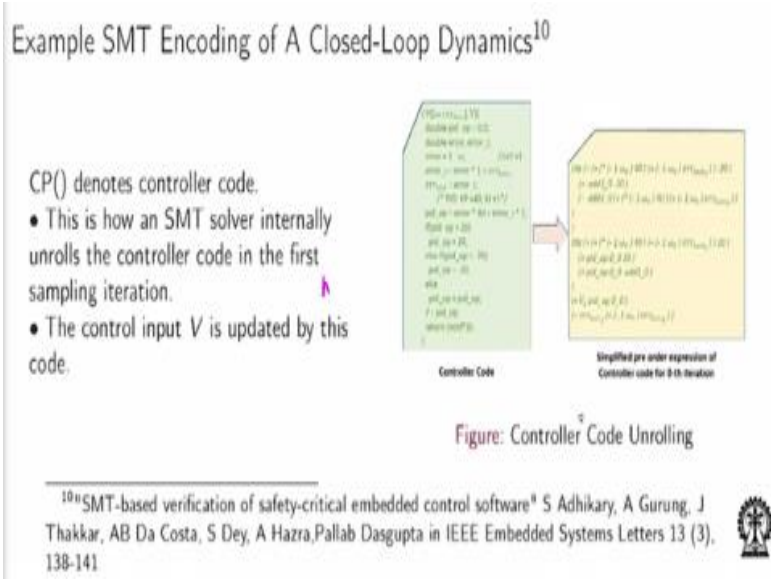
Now if there exist value assignments for which at certain points the safety envelope is going to be I mean violated those points we can the satisfiability solver can actually give you value assignments to this formula such that if I drive this system with those value assignments, I will see that the safety envelope getting breached. So, that is a nice example here.

(Refer Slide Time: 38:12)



So, we have a small example we consider a DC motor control system and this is a program which is representing the DC motor controller and here you have the automata the hybrid automata corresponding representing the dynamics of the DC motor. So, you can look at these things. So, you have a PI controller whose goal is to meet the plant's output reference like is given as one rad per second. And you also have this variables gt and lt there are two variables denoting the global time and local time of the system and the sampling period is maintained like this here. So, this is a program for the controller.

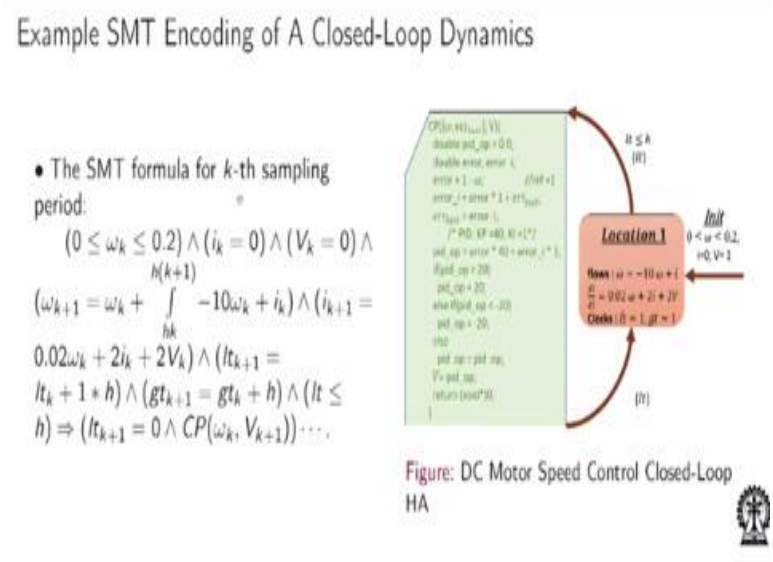
(Refer Slide Time: 38:56)



So, what will happen is this controller code just like we saw that whenever we have to handle updates to a variable, we rename it. So, same thing will happen for the controller code it will be transformed into a single static assignment version. That means for each variable multiple variants will be created whenever those variables are written. So, you see pid op, pid op will have values like pid op dot 0, pid op dot this that etcetera.

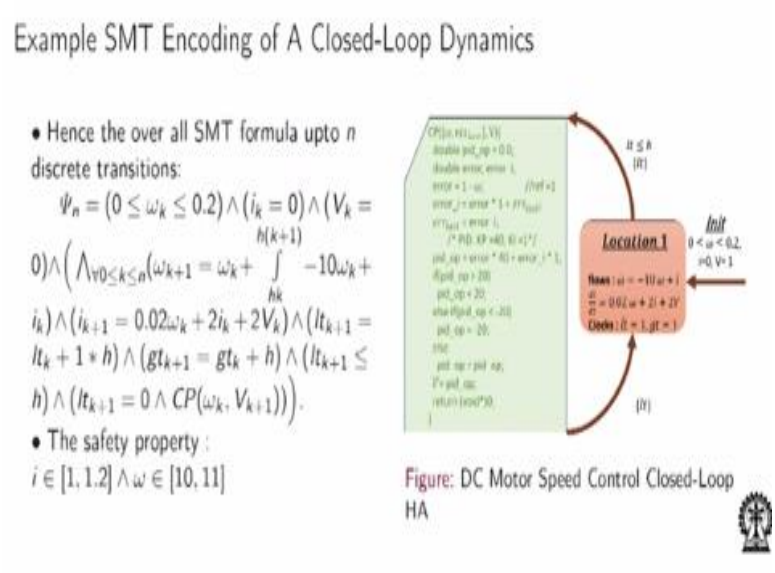
So, multiple such for each iteration this variable will keep on getting rename and also if you see this is written in a pre-order expression. That means operator followed by operand 1, operand 2. So, that is the format that is kind of used by most SAT solvers is typically this for standard SMT solvers. And I mean you need not be worried this is not to be done by the user just needs to give a program like this is the solver which will first or the front end of the server which will generate this representation which is to be used by the backend. So, it will be unroll the controller code up to those k number of iterations and then it will put it in a necessary form and then it will transform it into this pre-ordered expressions that means for every expression the operator will come first operand 1 and then operand 2. And then this is how the solver will modify the program. And then it will try to generate suitable SAT clauses for this program which we kind of abstractly represented here using this CP.

(Refer Slide Time: 40:33)



So, finally you will have an SMT formula for up to some kth sampling period which will look like this. So you can just check it out later on. We will be we are just showing it as an example and we also have a tool for this setup of how to check controller and a plant combination in closed loop for desired properties. So, there is a website where that tool is hosted and you can find out that tool being I mean that website referred to inside this paper.

(Refer Slide Time: 41:05)



And so that is how we SMT we do the encoding of a closed loop in this way. So, but again if you do if you see this is a single mode hybrid automata and there is a controller here so, for the single location you have the dynamics of the motor and then it is kind of periodically it is being interrupted and the variables by of the control are getting updated using this controller. And this closed loop is containing combined for n number of iterations, that means in each iteration you have suitable clauses that are getting added up into the formula.

(Refer Slide Time: 41:45)

Generalization of SMT formulation for HA

- Consider the hybrid system model $\mathcal{M}$ is modeled by an HA $H = (Q, X, f, Init, Inv, E, G, R)$.
- $Q = \{q_0, \cdots q_{m-1}\}$, $X_{i,q}$ denotes the valuations of X after i-th transition if at q location and the time interval spent in destination after a transition $I_i = [\tau_i, \tau_i']$
- The final SMT formula to verify here is:

$$\exists X_{0,q_0}, X_{1,q_0} \cdots X_{n,q_{m-2}} \cdots X_{n,q_{m-1}} \bigvee_{q \in Q} \Big(Init(q, X_{0,q}) \wedge (X_{i,q} = X_{i-1,q} + \int_{\tau_i}^{\tau_i'} f(X,q)dt) \wedge \forall \tau \in [\tau, \tau']$$

$$(X_{i,q} = X_{i-1,q} + \int_{\tau_i}^{\tau} f(X,q)dt) \Rightarrow Inv(q, X_{i,q})\Big) \bigwedge_{i=1}^{n} \Big(\bigvee_{q,q' \in Q} \Big(G(q, X_{i,q}, X_{i,q'}, q') \wedge (X_{i,q'} = X_{i-1,q'} +$$

$$\int_{\tau_i}^{\tau_i'} f(X,q')dt) \wedge \forall \tau \in [\tau, \tau'](X_{i,q'} = X_{i-1,q'} + \int_{\tau_i}^{\tau} f(X,q')dt) \Rightarrow Inv(q', X_{i,q'}))\Big) \wedge \Big(\bigwedge_{q \in Q} \Phi(X_{i,q}) \Big)$$

So, that is how it works generally now if you want to generalize this so this is a scary slide but it is actually easy to understand. So, all we are doing here we are now saying that well let us now consider a hybrid automata with many locations why there should be one location. So, you see the idea is very simple it is almost an extension building over that previous structure where we introduce only one location.

You will now have the same structure but you will have an OR-ing done over all the states which are initial states and then you will have the rest of the updates almost looking same. Again, you have this AND over n number of transitions from coming starting from here to here as you can see. And then inside this just to handle possible switches that the automaton can have from one location to another you will have an OR clause.

You see you can just blindly write this formula for all combination of this locations of the hybrid automata because as long as the transition relation does not hold the infeasible transitions will not be taken. And only when a transition is really there then the guard would be satisfied the transition relation would be satisfied and accordingly the update will happen following this formula. And then well the invariant will be satisfied etcetera.

So, you see that here the guard has to be satisfied. Then in the target state you go and you are going to satisfy you are going to get your variable updated and then after this update happens you need to you will need to satisfy the invariant. So, all you are doing is instead of having one clause for a single state you are having an OR for all possible state pair combinations and this will be evaluated only based on whether there is a feasible transition or not.

So, that will be also captured using suitable constants here. So, that is how it typically works there are lot of examples using the solvers if one is interested in. So, our idea here was that we will be introducing you to this concept of modelling simple hybrid automata and programs together using SMT solvers. But again let us remember this is just an introductory modelling do not get too scared out of this kind of complex modelling as it looks.

I mean for our tutorials we will have simple problems from this, where we saw that well for simple automaton and co program combinations how such SMT encodings can be done and we will be showing examples using such SMT encoding that how they can be how I can actually create a combination of a simple hybrid automata and the C program which is the controller and how we can write a formula which will represent all possible states that the system can visit the reach set starting from a set of initial locations.

We will do that as an exercise and then we will also show that how that can be applied to an SMT solver in the tutorials and what is the solver expected to return to you. So, I think with those examples once you understand the basics here, we are not covering everything we are just trying to introduce the topic because it is a big jump here which we fairly understand. I mean you do not need to memorize this thing first of all.

All you need to understand is the logic based on which we are trying to build this single state formula. You need to satisfy initial location and you need to satisfy invariant, you need to satisfy flow, you need to satisfy guard, you need to satisfy reset and you need to satisfy the control programs constraints. That is all we need to remember. When we will have tutorials, we will show through simple examples how this thing is really applied in real life. Thanks for your attention. We will end this topic here, thank you.