**Foundation of Cyber Physical Systems**

**Prof. Soumyajit Dey**

**Department of Computer Science and Engineering**

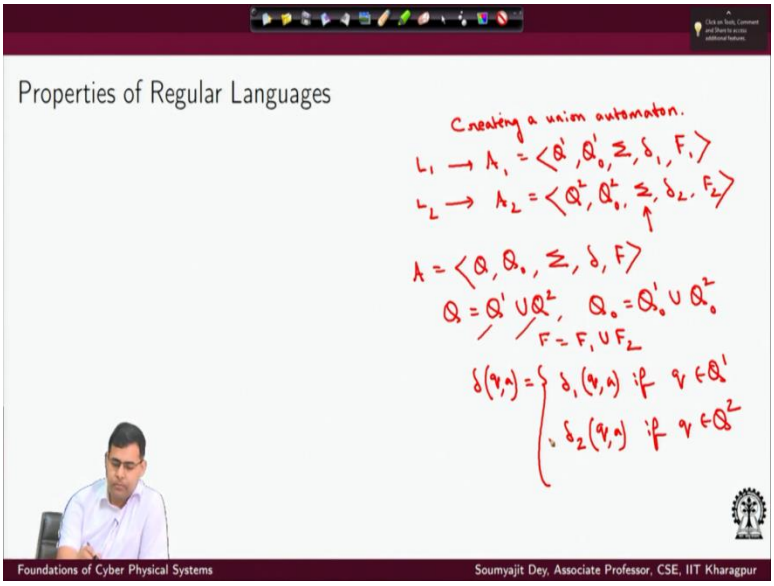
**Indian Institute of Technology, Kharagpur**

**Lecture - 29**

**Hybrid Automata Based Modeling of CPS (Continued)**

Hello and welcome to this course on Foundations of Cyber Physical Systems. So, let us move on with our lecture.

**(Refer Slide Time: 00:38)**




So, we have been talking about these properties of regular languages and we have discussed that under what operations the set of regular languages even closed etcetera. So, now we must define the corresponding algebraic operations on which this I mean how I can really generate language which or a corresponding automaton which is a union I mean which whose language is basically the union of the two input languages or similarly the intersection of the two input languages.

So, let us see about that. So, let us create a union automaton first. So, let us take the two regular languages $L_1$ and $L_2$ and for them we have these automatons that exist and their tuples are given by

$$L_1 = \langle Q^1, Q_0^1, \Sigma, \delta_1, F_1 \rangle$$

$$L_2 = \langle Q^2, Q_0^2, \Sigma, \delta_2, F_2 \rangle$$

So as you can see that we are assuming that of course the alphabets are same here. Now it can be easily shown that the union of these two languages can be accepted by an automaton A for whom the tuples can be defined like this.

So, if we want to define the transition function of this automaton, it is something like this.

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & if\ q \in Q^1 \\ \delta_2(q, a), & if\ q \in Q^2 \end{cases}$$

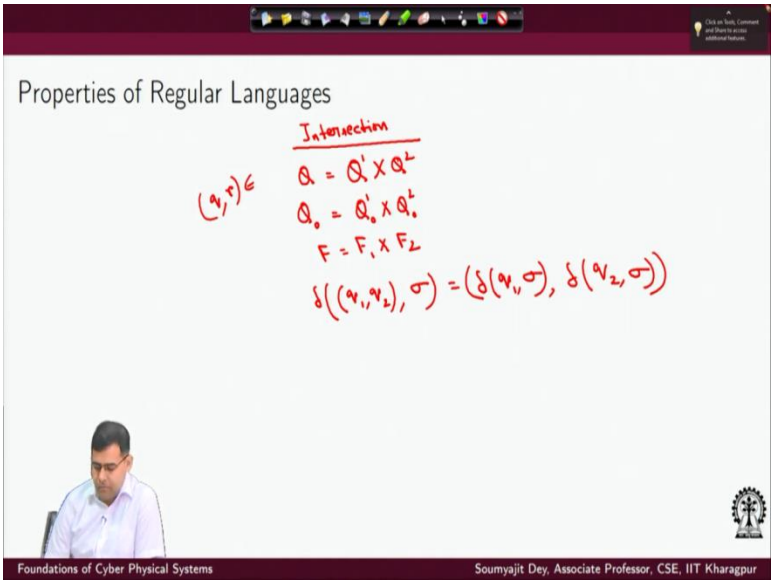
So, let us see what we have just done here. So, we are saying that the final automaton will have its effective set of states which is nothing but the union of the set of states of the original to automaton. The set of initial states will also be nothing but the union of the set of initial states of the original two automaton.

Similarly, for the set of final states and the transition function is also going to be like this, that its either going to. So, this fine this bigger automata the union automaton will be changing states either using the first transition function, that is the component one I mean a one's transition function or a two's transition function based on I mean which state it is currently in, if it is a state it is a member of these or if it is a state which is a member of this.

So, I think that is I mean that is very simple to understand that what essential is happening. Basically we are combining this automaton their stairs their final states and their transition function everything. So, essentially that will give the I mean that automatically will create an automaton

which will accept kind of any string which is accepted by any one of this automaton. Now there is the union case, now let us look at the intersection case.

**(Refer Slide Time: 05:43)**

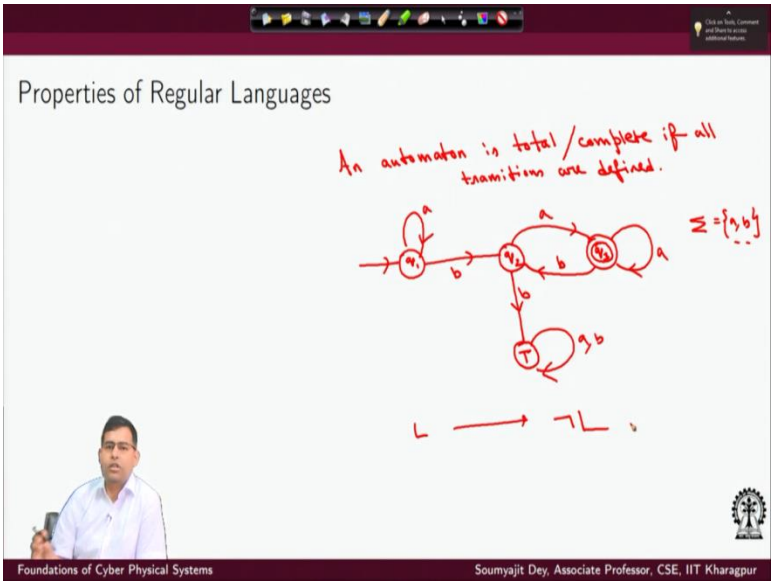



So, for creating this automaton for the intersection what we will first do is we'll create a Cartesian product of the two state spaces, we will create a Cartesian product of these two state spaces of the in two component automatons. So, essentially the set of states here will be given by this product that means the states will be like tuples. So if you have state q from com automaton 1 and if you have a state r from automata 2.

So, this (q,r) is like member of this set. Similarly the set of initial states. And if you look at the transition function now. So, it is now operating on this kind of state tuples, so this is a state tuple on which the transition function is operating and let there be an input event sigma. So it will just follow both the transition functions. So it is like this. So, using sigma wherever I go from $q_1$ and using sigma from wherever I go from $q_2$ both of them will be followed.

So, that is how the intersection would be defined. Now what about complement? I think before going into complement, we will need to define what does it mean by your automated description being complete or what does it mean by an automaton description being total.

**(Refer Slide Time: 08:19)**




So, let us draw an example. So, this is the definition. If I mean all the transitions that are possible in the automaton have to be defined. So, we take this example automaton here. So this is your initial state with the self-loop let us say. So let us observe this automaton for a while. Are all the transitions defined. So, if your alphabet is this a and b there are only two symbols in this alphabet. That means the requirement is that from every state there should be a defined transition following all the members of this alphabet set.

So, from $q_1$ with $a_i$ go to $q_1$ only, from $q_1$ with $b_i$ go to $q_2$, from $q_2$ with $a_i$ go to $q_3$, but from $q_3$ with $a_i$ go to $q_3$, from $q_3$ with $b_i$ go to $q_2$, but from $q_2$ with the input event b where do I go that is not defined. So, this is not a kind of total or complete automaton. So, how do I make it total? So, for all such cases you can create an equivalent automaton, equivalent in the sense that the language will be equivalent, by just adding a transition like this two an extra state which we call as a dead state or a terminal state.

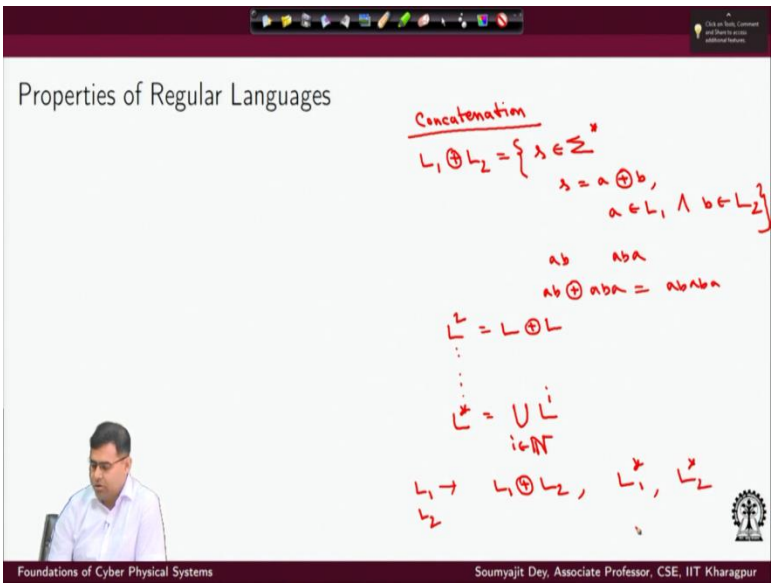
That means since the original automata was not supposed to go anywhere with b. So let it if the b happens in this state let it go to this kind of a terminal or a dummy state here and then the automata should be stuck here, that means, with a whatever input comes it is not going to be anywhere. So,

that is how you create a complete version of this automaton that means all the inputs and etcetera are defined all the possible transitions are defined.

Now for any such complete automaton if it has a language L, we can define the corresponding language complement of L by a very simple transformation. The transformation is just complement the state of final states that means whatever is the current final state should not be the final state and whatever else other set of states are there should all be final states. Now how does it help?

What is the complement of the language that means whatever strings lead to accept they would not be accepted. And all possible other strings in sigma in the entire universe of the language must be accepted. So, that is what, that means you make everybody else as final stats without the current final step. So, that is how you define a complement language.

**(Refer Slide Time: 12:21)**




Now just to recall I mean just to remind everybody that the reason we are talking about languages their classes and the importance in the context of CPS is that like we saw that regular systems regular discrete systems like vending machine like maybe some other system they can be represented by this kind of automaton. And if we know the properties of automatons, we also know

that which other complex system which can be broken down into this kind of simple systems is also a finite automaton.

Or how I can build a larger model of finite discrete system using some smaller models. So, these all we are able to connect with this language theory and system modelling by doing this part here. Now just another operation we will like to define here, another operation on this proper on this set of regular languages and with this we should be done with this specific part here. So, let us take up two languages $L_1$ and $L_2$ and we define their concatenation as this.

$$L_1 \oplus L_2 = \{s \in \Sigma^*, s = a \oplus b, a \in L_1 \wedge b \in L_2\}$$

So, it is a set of strings where every string is further a concatenation of two component strings where the first component is from $L_1$ and the second component is from $L_2$. So, as you can see that this operation, we are lifting this concatenation operation we are actually lifting from strings to language, and we are saying that languages can be concatenated just like strings can be concatenated.

For example, let us say ab is a string and aba is a string. So concatenation of this is nothing but we just write them together we just teach them together similarly we can switch languages together. Now why is this important? Because think of concatenator language with itself, so L square I can define L square as nothing but L concatenation L. And this gives us in this way I can define L cube, $L_4$, like that and finally L star.

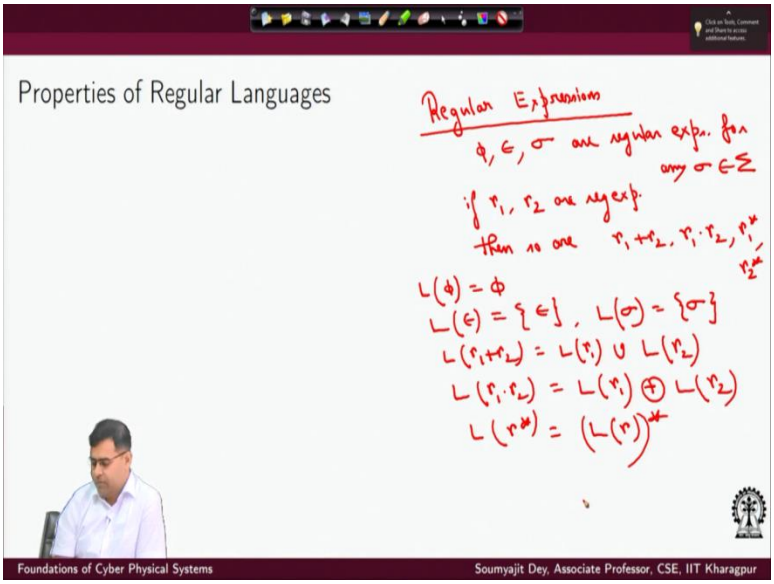
Or the closer the clean closer of the language like this,

$$L^* = \bigcup_{i \in \mathbb{N}} L^i$$

set of naturals L to the power i that means all possible powers of L you take the union. So, for any language you can define its screen closure like this. Now why is this very important? Because given any language L I mean operations like I mean you can have concatenation of I mean peak

two languages $L_1$ and $L_2$ they are concatenation is also is a regular language. Not only that they are clean closures $L_1^*$, $L_2^*$ they are also these regular languages.

**(Refer Slide Time: 16:15)**




Now in this context there is another important concept which is known as regular expressions. So that is just another way to represent regular languages in a succinct form. Let us see that. So, what are really regular expressions? So, if we have to define what is a regular expression it should be some, we will start it like this. So, it is a definition which goes like this that the nulls, the null set of strings, the empty string, and any, I mean all strings made with singleton inputs from the alphabet, they are regular expressions, they are all regular expressions in their own capabilities. For any or any symbol in the alphabet that symbol itself is a regular expression the empty string is a regular expression, the null string is also a regular expression. And inductively we can say that if we pick up any two regular expressions $r_1$ and $r_2$, then $r_1 + r_2$, $r_1$ concatenation $r_2$ and $r_1$ star $r_2$ star they are all regular expressions.

Now of course what does this mean this simply means $r_1 + r_2$ simply means the language of $r_1$ union language of $r_2$ that is the corresponding language. So, I can just say something like this I can just take a language theoretic view here and define all these things. So, the language here of this regular expression simply is the null set. The language corresponding to say every regular expression corresponds to a language which is a regular language.

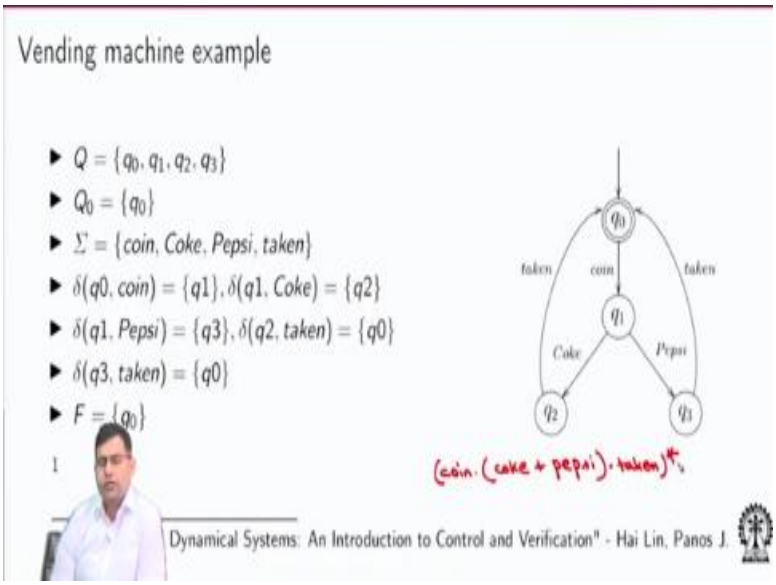
So, this is nothing but a language containing only the empty string, this is the empty string. The language corresponding to a regular expression which has only one symbol is nothing but a singleton set with that symbol. Language for $r_1 + r_2$ is nothing but the language for the regular expression $r_1$ union the language of the regular expression $r_2$ language of this is now, this is where our previous definition will come in.

So, if you can see that the previous definitions tell you how to expand a regular expression to a language and here you are just, I mean defining these operations with concatenation like we define earlier and language of some regular expression r*. So, this is how r star is defined. So this definition is in terms of the language itself, so you will have you just take what is the language of r and star over there apply the star operation over there.

So, this is how we link up we define regular expressions in this in this recursive manner. The basic regular expressions and their combinations using $r_1$ dot dot to $r_2$*, $r_2$*, $r_1 + r_2$ like that. And we use this there I mean we see their relations with the corresponding language definitions that how a regular expression is defined or operated upon in using the operations that we have defined on the regular sets.

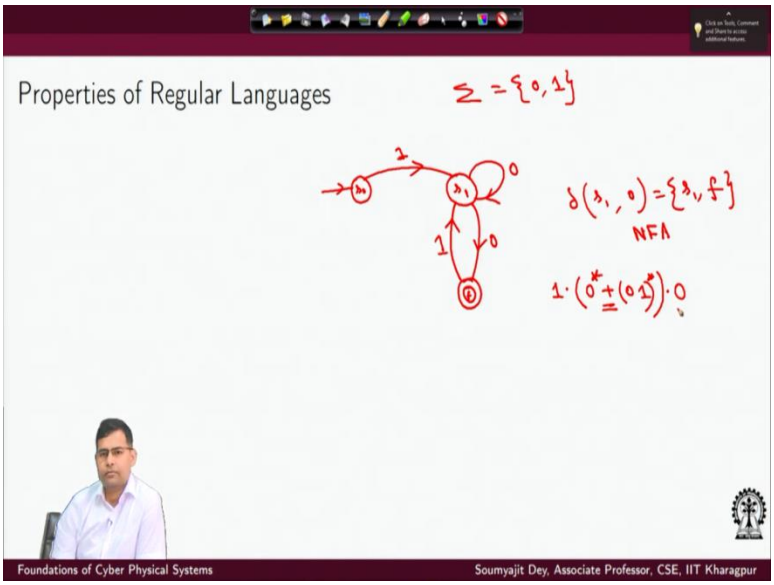
So, if we use these notations of regular expressions that means the notations, we have introduced are plus dot plus connects with at the language theoretic union, dot connects with the language theory concatenation and star is the language the clean closure of the language.

**(Refer Slide Time: 21:06)**




And if you use these notations to define what is the language of the vending machine automaton, if you see it will be something like this is kind of obvious. So, you have a coin and you go coke plus Pepsi there are two choices here and then there is only one choice taken and there is a loop, so it will generate a clean closer and you will have something like this.

**(Refer Slide Time: 22:13)**




So, will end this discussion with maybe one more example here. So, let us take another sample. So, here let us assume we have a Boolean alphabet so $s_0$ is 1 this is like an automaton here and as you can see is this DFA or an NFA well from s1 you have two possible states. So, I can just write

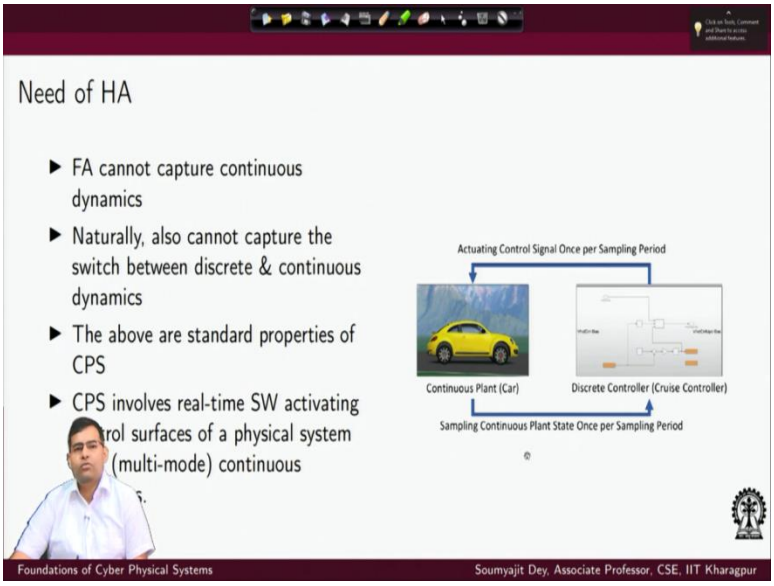
delta $s_1$ with 0 you can land up in $s_1$ or you can land up in f both are possible. So, it is definitely an NFA here.

Now can I create a regular expression for the language of this automaton? Of course, yes, so I have asked you have a one. So any string that is accepted by the language must start with a one. Now after the one if you see you have two choices here you can loop in here or you can loop in here, so you can non-deterministically select whether to loop here or whether to loop here. You see this loop is also kind of inhabitable.

I mean because you can just come here and stop or you can just take this loop and then take this loop or this loop again. But eventually you must take this 0 to get to the final step. So, it is certain that you start with the one here and you end with the zero here. For any string that is to be accepted these two things must be the case and in between you can either loop here from s1 or you can to or, so you have this choice here, you can just execute this loop that is it.

So, start with the one have a loop here or have a 0 1 loop here and eventually have this 0 at the end. So, that is how this automata is going to operate.

**(Refer Slide Time: 26:05)**




So, this is about our treatment of regular languages and automaton. But the eventually we want to study what we have called as hybrid automata and here we will be just trying to motivate why hybrid automata is required. So, as we have seen that simple discrete systems with discrete states and jumps between discrete states can be modelled using finite automaton. But what is important is finite automaton is unable to capture any kind of continuous dynamics which is kind of a very important property.

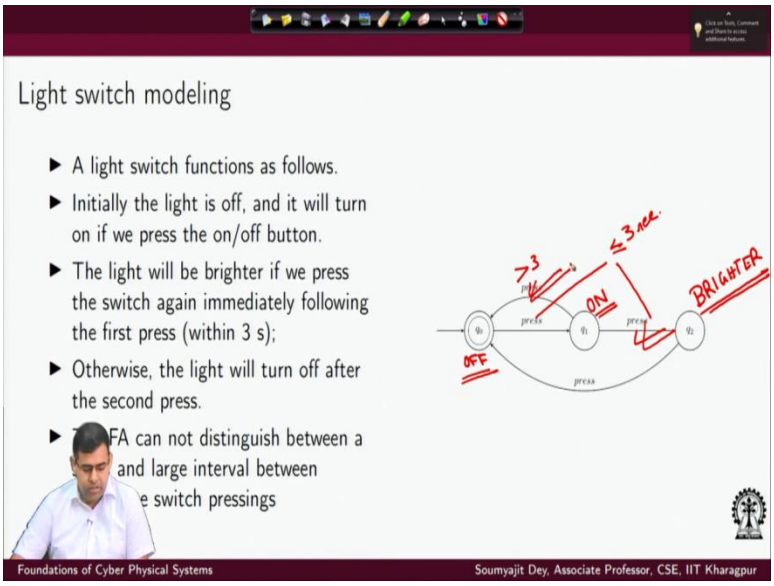
I mean any hybrid automata and any Cyber Physical System for that matter may have some continuous dynamics. So, since finite automata is only about discrete state switches it cannot capture continuous dynamics. So naturally it cannot also capture I mean this kind of switching that may happen between a discrete and continuous dynamics. Now this above thing this discrete and continuous dynamics and their switch is like a standard property of Cyber Physical Systems.

Why? Because every Cyber Physical System will have some real time software. So that is the discrete part and that would activate the control surface of a physical system which is have some continuous dynamics. So, let us say you have an antilock brake or some other system like a cruise controller, so it will be taking some discrete decisions and those decisions are taken by the software.

So, these are the logical changes that are happening to the system and they will be manifested in terms of a control input and based on the control input the car's dynamics will evolve here. So, that is a continuous evolution, so you will have discrete state changes here. So it is like here you have an automata based model of a program where the system is jumping among states and it is kind of calculating control inputs for this and then there is a continuous evolution.

After that there is signal, I mean a measurement sampled here and again this automaton is going to take some decision. I can think of it like this, now that would automatically mean that I need a system where I need a finite or I need a formal model where both these things can be captured which unfortunately, we cannot do using a finite automata.

**(Refer Slide Time: 28:41)**




So, just to get into the hang of the situation to pinpoint to exactly what is the problem with this, let us take another example. We want to model a light switch. So, this is also an example taking from the same book which we talked about earlier from where we took the vending machines example. So, here we have the light switches specification written. The specification is very simple this is the English language specification initially the switch is off.

So, q0 is a state which says that the switch is off. So, in fact let us write it here. So this is the off state and if we press this button then the switch then the light is turned on here and if we press that button immediately again then the light will become brighter. So, the light is brighter here but for the light to be made bright it is essential that the switch is pressed twice very quickly starting from the offset immediately after you make it on you must press again.

So, that means these two presses these two events must not be separated by I mean more than 3 seconds these two events should occur inside 3 seconds for this to happen and if these things differ by more than 3 second then the light actually turns off it does not become bright. Now clearly as you can see that this is a requirement which this finite automaton does not understand, I mean we are unable to model it.

This automata model that we have cannot capture this intricacy of time, calculation over time, I mean, difference of time etcetera. It just has this press event; it is unable to distinguish between this press event and this press event by figuring out what is the amount of real time in which this press events differ. I mean because if the press event occurs within a short time interval, then it is this press event the automaton goes this way.

If the successive press events occur with a bigger time interval, then 3 second, then it is actually taking this path this press event. So, that is the specific issue, the finite automata cannot distinguish between the small and large interval of the successive switch pressings and that that makes this model not very useful in this context. So, this tells us that what we desire to be in a hybrid automaton.

So, in our next lecture will be introducing this formal tuple of hybrid automaton and we will be talking about how this specific example can be modelled as a hybrid automaton. Thank you for your attention.