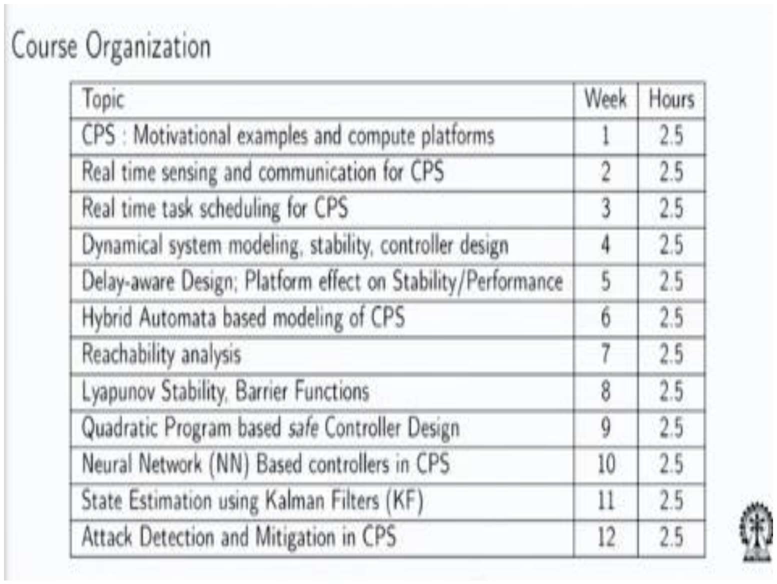
**Foundation of Cyber Physical Systems**
**Prof. Soumyajit Dey**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kharagpur**

# Lecture - 28
# Hybrid Automata Based Modeling of CPS

Hello, welcome to the course. Today we will be talking about the topic in lecture 6 primarily week 6 topics.

**(Refer Slide Time: 00:39)**

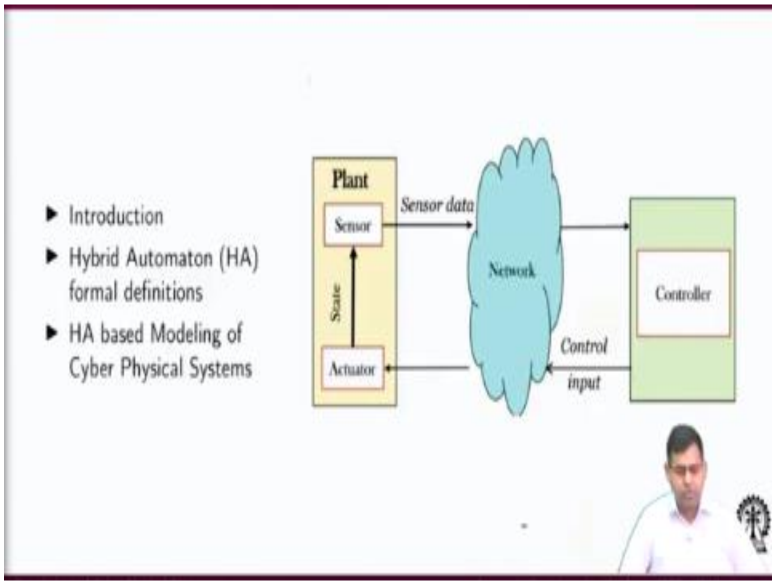


## Course Organization

| Topic | Week | Hours |
|---|---|---|
| CPS : Motivational examples and compute platforms | 1 | 2.5 |
| Real time sensing and communication for CPS | 2 | 2.5 |
| Real time task scheduling for CPS | 3 | 2.5 |
| Dynamical system modeling, stability, controller design | 4 | 2.5 |
| Delay-aware Design; Platform effect on Stability/Performance | 5 | 2.5 |
| Hybrid Automata based modeling of CPS | 6 | 2.5 |
| Reachability analysis | 7 | 2.5 |
| Lyapunov Stability, Barrier Functions | 8 | 2.5 |
| Quadratic Program based *safe* Controller Design | 9 | 2.5 |
| Neural Network (NN) Based controllers in CPS | 10 | 2.5 |
| State Estimation using Kalman Filters (KF) | 11 | 2.5 |
| Attack Detection and Mitigation in CPS | 12 | 2.5 |

So, what we have new in this week is we will be talking more on modelling aspects of Cyber Physical Systems from a formal perspective. So, let us understand what it means.

**(Refer Slide Time: 00:51)**

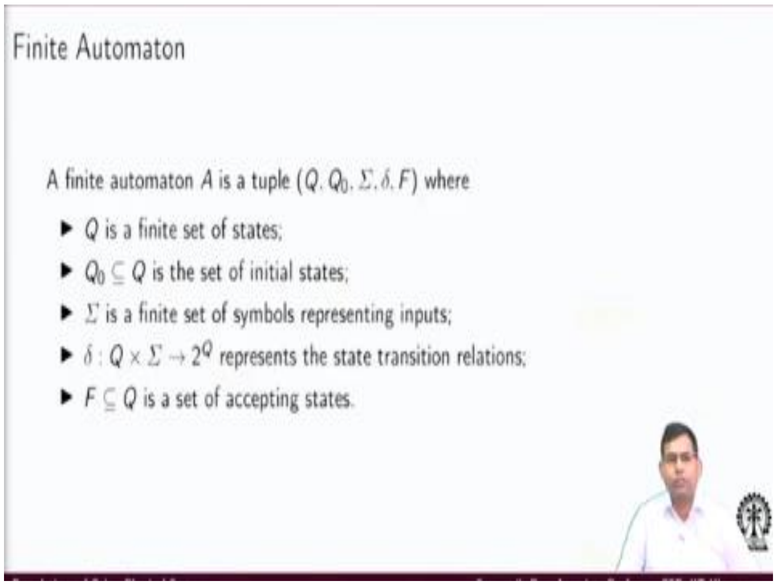


- Introduction
- Hybrid Automaton (HA) formal definitions
- HA based Modeling of Cyber Physical Systems

So, we will be talking about automata theoretic Foundations of Cyber Physical Systems. Now the one may argue that why is that required because for analyzing Cyber Physical Systems you will need to have some method through which you can create a formal model which is both simulatable as well as it has the mathematical recurve in inherent in it is in the way that it has been built, so that you can do some formal app for some application of formal methods through which you can guarantee certain properties of the system. And as we all know that these are safety critical systems so we need to figure out that we need to actually have methods through which we can actually claim that these systems are going to be safe, they are never going to create an unsafe region stuff like that.

Now one way for doing that would be applying formal methods and also simulating them with lot of inputs. Now in both cases for simulation as well as applying some formal logic-based reasoning method, we need a sound semantic foundation in the modelling paradigm of the Cyber Physical System. So, what we will be touching upon is hybrid automaton-based models. Now hybrid automaton is kind of advanced automotive theoretic representation which is definitely required for Cyber Physical Systems. So, let us let us understand what it really means.

**(Refer Slide Time: 02:29)**



## Finite Automaton

A finite automaton $A$ is a tuple $(Q, Q_0, \Sigma, \delta, F)$ where

- $Q$ is a finite set of states;
- $Q_0 \subseteq Q$ is the set of initial states;
- $\Sigma$ is a finite set of symbols representing inputs;
- $\delta : Q \times \Sigma \to 2^Q$ represents the state transition relations;
- $F \subseteq Q$ is a set of accepting states.

So, before getting into hybrid automaton, we must understand that what really a finite automaton use. This is something that some people may be already aware of but of course this is an interdisciplinary subject and looking at the general audience we start from the basics of finite automaton here, what it means by a finite automaton. So, a finite automaton is a mathematical structure which is primarily used to model systems where the number of states is going to be finite.

That is why basically a finite state automaton. Now they have got their usage in several branches of engineering. In computer science we have we see lot of usage of finite automaton or finite state machines in the specific context of digital circuit design, synthesis techniques etcetera. And of course, in several other branches we have got this kind of finite automaton-based modelling of systems it is used for representing a large class of systems.

So, let us first understand this mathematical tuple. So, overall, a finite automaton is given by this kind of a tuple with five entities. So, what we have here is the first entity which is nothing but the set of states. Suppose I am trying to model a system. So, what are the different states in which the system can be located at any point of time so that is like a finite set of states. And a subset of that is Q0 which we call as the initial states.

That means that these are the states from which the system may start and then they may be evolving to some other state etcetera. Now what we have next is sigma which is a finite set of symbols representing inputs. So, essentially you have a finite a set of symbols which kind of trigger the transition of states from one state to another state and this set of symbols is known as the alphabet of the automaton.

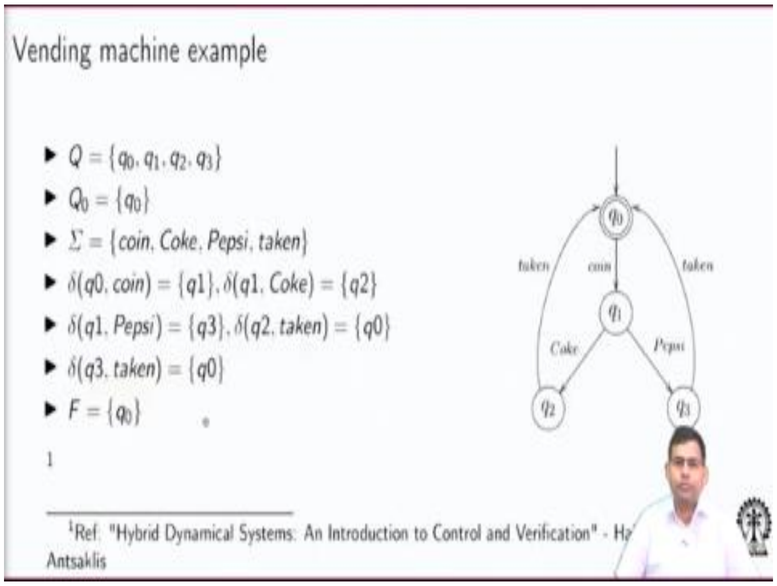
Essentially this is the standard symbol which is used in most books. Now we come to the next thing which is what we call as the transition function. So, delta represents a transition function which actually tells you given a state and given an input, on which state possibly the system might change over to. Now as we can see that on the right-hand side, we have this 2 to the power Q instead of having just Q. That means well this is used to have two possible meanings here.

I mean it this is used to represent two classes of automaton and this is the most general representation. So, once we write 2 to the power Q, that means we are talking about the power set of states, which means in the general case given a state and an input we are saying the system can go to a set of possible locations or set of possible states. Now so that is what it means when in the right-hand side that is the range set is represented by this power set of the set of states.

Now had this been Q, that means for every state and every input we would be giving only one choice to the system. So that is a deterministic automaton. And once we make it as 2 to the power Q, we are saying that given a state and an input, the system can jump to a set of possible states any one of them. So, this brings in the notion of non-determination in the system and that is what we call as a non-deterministic finite automaton.

And finally, what we have is a set of accepting states F is a subset of Q that means if the systems run is getting into any of these five accept states, then we say that we have a sequence of inputs for which is accepted by this automaton. We will see more into that in detail.

**(Refer Slide Time: 06:39)**



## Vending machine example

- $Q = \{q_0, q_1, q_2, q_3\}$
- $Q_0 = \{q_0\}$
- $\Sigma = \{coin, Coke, Pepsi, taken\}$
- $\delta(q0, coin) = \{q1\}, \delta(q1, Coke) = \{q2\}$
- $\delta(q1, Pepsi) = \{q3\}, \delta(q2, taken) = \{q0\}$
- $\delta(q3, taken) = \{q0\}$
- $F = \{q_0\}$

1

[1]Ref. "Hybrid Dynamical Systems: An Introduction to Control and Verification" - H
Antsaklis

So, let us take some examples here. So, what we have is an example of a vending machine. So, let us see what a vending machine is. So, well it could have been modelled in many possible ways. This is just one possible simple modelling of a vending machine. So, we are saying that if the machine which can be in any of these four states. And so, when I represent a state with an incoming arrow from outside so that is like what we call as the initial state of the system.

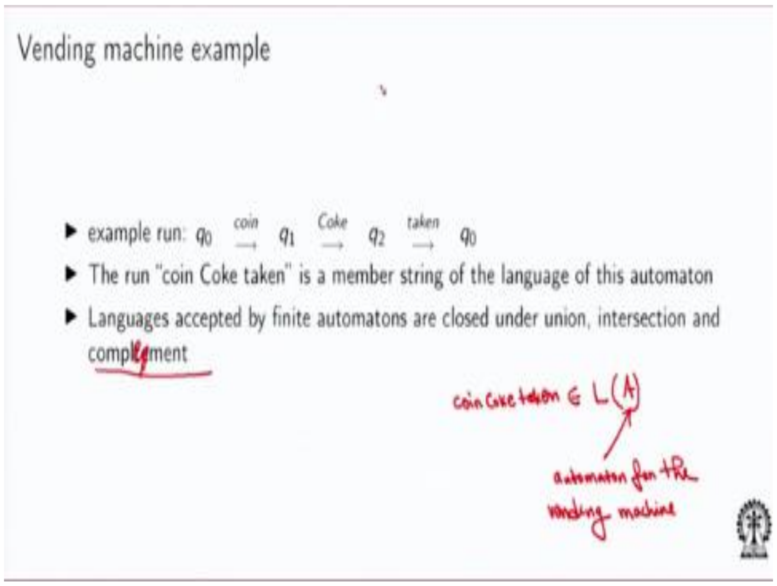
And also, if I represent a set with two concentric circles that is typically the symbol which we will use to represent a final state. In this case it happens to be that both the initial and the final state are same. So, this is where the vending machine starts. If an input in the form of a coin is provided it goes to another state where it will ask for a choice from the user. Whether the user wants a coke or a Pepsi based on whatever the user inputs it goes to either this side and dispenses a coke.

So, it will and once the user takes the coke from the vending machine so that fires this event taken. So, you can see that these are all user inputs. The user inputs are driving the machine, the user gives a coin, the user gives a choice of coke, the user, the system goes to the state here where it will dispense a coke and when the user picks up that so that is the last event taken and the system is reset to its original state.

So, as you can see through this example you have a finite automaton with these four states $q_0$, $q_1$, $q_2$, $q_3$ and $q_0$ is the initial state. And sigma is kind of the alphabet of the system coin, coke, Pepsi and taken. So, these are the four inputs that you have. And here we have our transition function. So, this transition function is kind of representing all these transition relations that we talked about that well if I am in $q_0$ and a coin comes in the system will jump to $q_1$.

If I am in $q_1$ and the event coke comes in the system jumps to $q_2$ and so and so forth. So, that is how it operates and we have this final state that is $q_0$.

**(Refer Slide Time: 09:04)**



## Vending machine example

- example run: $q_0 \xrightarrow{coin} q_1 \xrightarrow{Coke} q_2 \xrightarrow{taken} q_0$
- The run "coin Coke taken" is a member string of the language of this automaton
- Languages accepted by finite automatons are closed under union, intersection and complement

coin Coke taken $\in L(A)$

automaton for the vending machine

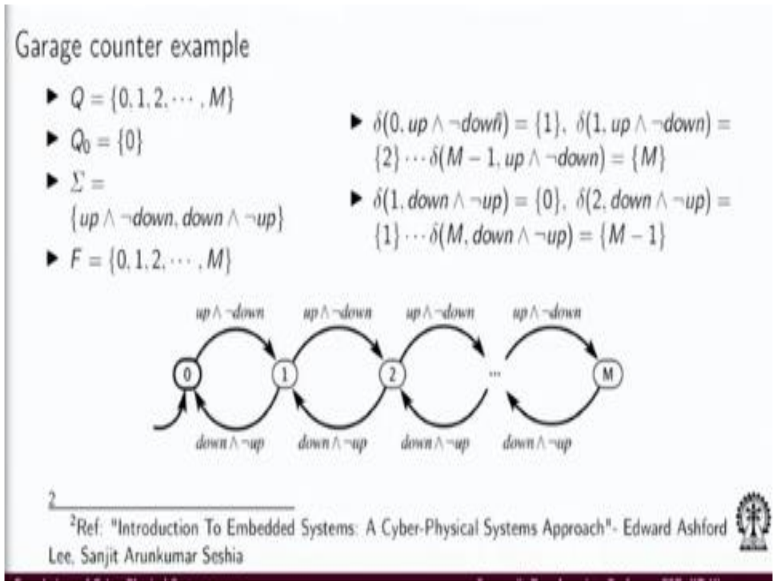
So, here we have this systems example with a run where we start at $q_0$ and then we go to $q_1$ then $q_2$ and then again $q_0$ and that is how the system works. So, as we can see that in this run this sequence of inputs that is coin, coke and taken. So, this sequence of inputs taken from the alphabet they form a member string of the language of the automaton. Why? Because with this string the system could start from the initial state and the system could reach the final state.

In this case both the initial and the final state happen to be the state $q_0$. Now coming to another important property that what do I mean by the language of the automaton. Now as we said that language of the automaton is something I mean for example this is the candidate string of that language. That means in general we can say that any string which takes the system to the initial, from the initial state to any final state is inside the language of the automaton, and that kind of succinctly defines given any automation A, what is its language let us say LA. So, we can just define it like that. Or in symbolical terms, this should be written as like this string is a member of the language of some automaton A, where this A is the automaton for that vending machine that we have been talking about. So, overall, that is how for given any automaton we can define its language and this leads us to some interesting properties of regular languages or languages accepted by finite automatons.

We will I mean in general any automaton if a finite automaton if you can create any finite automaton the language that the automaton accepts is known as a regular language in the terms of the language formal language hierarchy in computer science. And it happens to be the case that such languages are closed under set operations like union, intersection and complement. So, that is the standard set operations that we know of.

And what does it mean by the language being closed under these operations. So, let us try and figure out what is the inner details inside this.

**(Refer Slide Time: 12:07)**



## Garage counter example

- $Q = \{0, 1, 2, \cdots, M\}$
- $Q_0 = \{0\}$
- $\Sigma = \{up \wedge \neg down, down \wedge \neg up\}$
- $F = \{0, 1, 2, \cdots, M\}$
- $\delta(0, up \wedge \neg down) = \{1\}$, $\delta(1, up \wedge \neg down) = \{2\} \cdots \delta(M-1, up \wedge \neg down) = \{M\}$
- $\delta(1, down \wedge \neg up) = \{0\}$, $\delta(2, down \wedge \neg up) = \{1\} \cdots \delta(M, down \wedge \neg up) = \{M-1\}$

$$\begin{tikzcd}
\arrow[r] & 0 \arrow[r, bend left, "up \wedge \neg down"] & 1 \arrow[r, bend left, "up \wedge \neg down"] \arrow[l, bend left, "down \wedge \neg up"] & 2 \arrow[r, bend left, "up \wedge \neg down"] \arrow[l, bend left, "down \wedge \neg up"] & \cdots \arrow[r, bend left, "up \wedge \neg down"] \arrow[l, bend left, "down \wedge \neg up"] & M \arrow[l, bend left, "down \wedge \neg up"]
\end{tikzcd}$$

[2] Ref: "Introduction To Embedded Systems: A Cyber-Physical Systems Approach"- Edward Ashford Lee, Sanjit Arunkumar Seshia

So, before that we will also discuss some more examples. For example, this is the garage counter. So, this is a simple example which has been taken from this reference book Introduction to Embedded Systems or CPS approach by Edward Lee and Sanjit Seshia. Also, I must say that this previous example was taken from this recent book Hybrid Dynamical Systems. So, we will actually inside this course have many examples and many definitions which have been almost borrowed from these books.

We will be explaining them but given that their definitions are quite distinct and perfectly done, this is the primary reference from where this these things have been taken. Now coming to this example so the reason we have this example here is something specific. So, you can see that we are saying that well let there be a garage counter that means it is simply counting the number of cars that are going inside a garage and the number of cars that are coming I mean some cars will be leaving the garage.

And in general, we have a counter that is maintaining that the number of cars that are really inside the garage and that is what is being counted here, that what is inside, how many are inside. So, whenever a car gets in the counter will get an up-count signal and whenever a car gets out the garage will get a down count signal. What we are trying to impress upon here is let us say that maximum capacity is from 0 to up to M. So, the counter is going to have M number of states.

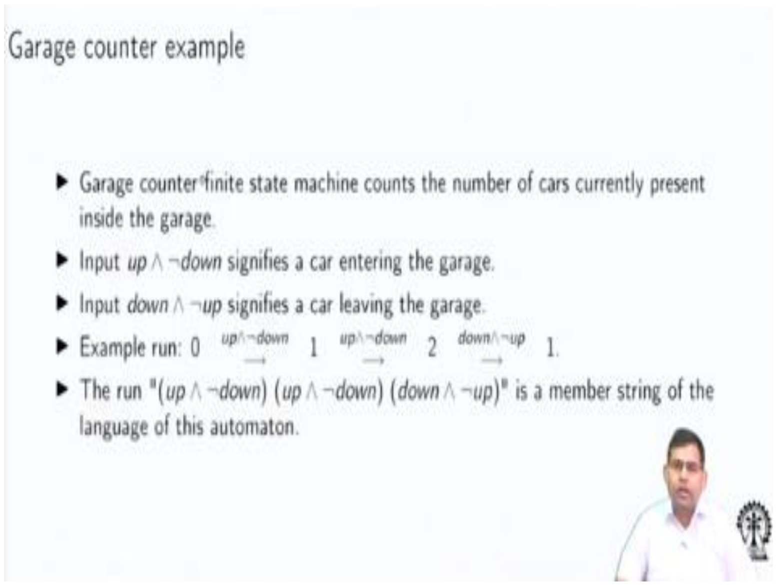
But what is interesting here if you see we are characterizing these events of that the count goes up or count goes down by different what we call as Boolean predicates up or down or propositions. And this will be events that either resolve to being true or false. So, basically these Boolean propositions and their Boolean combinations will also become eventually Boolean statements which are either true or false and we are kind of using them as transition cards inside this automaton.

So, the idea is that when I am leaving the count 0 and going to state 1, this is the event that must be true that the count the counter has two input lines. If you think of it physically, one is the command to go up another is the command to go down. It can never happen that up and down are both together one that is an inconsistent state. And similarly, so you have the only two these possibilities it is up and not down and it is down and not up.

So, these are the two consistent commands that the counter can have. So, in instead of having just an input event what we are having is some property about the system which is true. In this case the properties that the up value is 1. So, this Boolean predicate is 1, the down value is 0, so negation of down is 1 and they together is 1. So, this overall event is 1 it evaluates to true, logical true or logical 1. I would say and that is why this transition happens.

So, as you can see that this is something that is acting as like what we should call as transition guard. This is inhibiting the transition until this event becomes true and when this is true the guard is relaxed and you can transit from here to here. The point we are trying to make here is that not only it can happen that there is an input event, but you can also have this case that there is a situation on which the transition becomes possible, and that situation can be represented by a logical formula which will evaluate to true at some point of time, and at that point of time that the transition is taken. So, this is like a transition guard which may or may not be active and that is also dictating the transition here. So, that is what this specific example is trying to impress upon you.

**(Refer Slide Time: 16:25)**



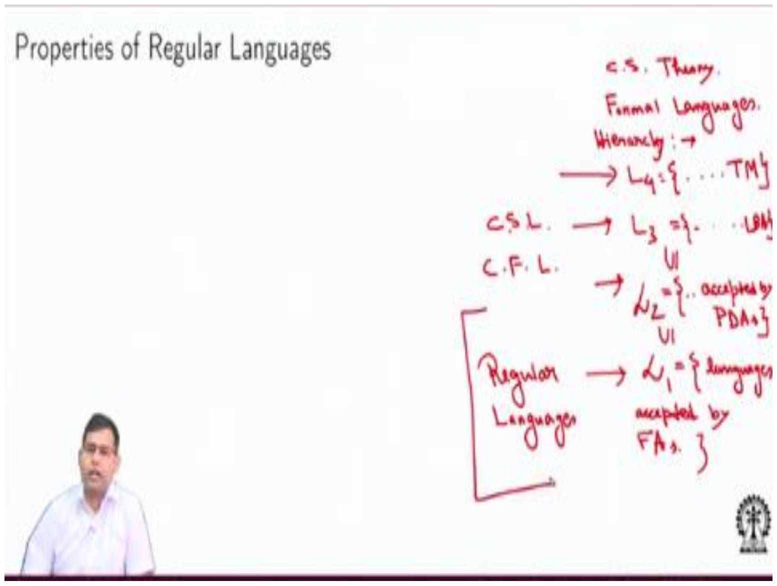
## Garage counter example

- Garage counter finite state machine counts the number of cars currently present inside the garage.
- Input $up \wedge \neg down$ signifies a car entering the garage.
- Input $down \wedge \neg up$ signifies a car leaving the garage.
- Example run: $0 \xrightarrow{up \wedge \neg down} 1 \xrightarrow{up \wedge \neg down} 2 \xrightarrow{down \wedge \neg up} 1$.
- The run "$(up \wedge \neg down)\ (up \wedge \neg down)\ (down \wedge \neg up)$" is a member string of the language of this automaton.

So, overall, this is the summary, that this is the finite state machine that counts the number of cars currently present inside the garage. So, you have events like up and negative down that signifies a car entering the garage, you have an input event like down and negative up, which signifies the car leaving the garage. Let us take an example run of this. So, if it is up and they negate down, you count up if it is down and negate up, you count down.

So, that is what is happening in this run. So, you can see that so this is primarily how this automaton is going to operate.

**(Refer Slide Time: 17:03)**


Properties of Regular Languages
C.S. Theory.
Formal Languages.
Hierarchy :→
→ L4 = { . . . TM }
C.S.L. → L3 = { . . . LBA }
C.F.L. → L2 = { .. accepted by PDAs }
Regular Languages → L1 = { languages accepted by FAs. }

Now coming to the other part that is properties of regular languages, that means like I said that any language that we have there is a kind of accepted by a finite automaton is a regular language. So, they have their properties, some of the properties they have already mentioned. So, let us give some time to the understanding of this kind of properties. So, before getting into the properties let us understand that in C.S. theory you have this hierarchy of languages.

So, in this hierarchy of languages at the lowest level you have the set of languages which are all accepted by, so you can say that these are languages that are accepted by finite automatons. So, this is what we call as regular languages. At a higher level we have languages that are accepted by what we call as push down automatons. So, what is a push down automaton? So, this is at the higher level. So a push down automaton is nothing but a finite automaton along with a stack.

So, it is just a normal stack that means it grows in one direction with a FIFO structure. So, in that way we can say that every finite automaton is also a push down automaton. I mean, that means because if every finite automaton there will exist a corresponding push-down automaton where the automata structure is same and there is a stack which is not doing anything. So, this language class actually subsumes the lower-level regular language class and this is known as what we call as the context-free languages.

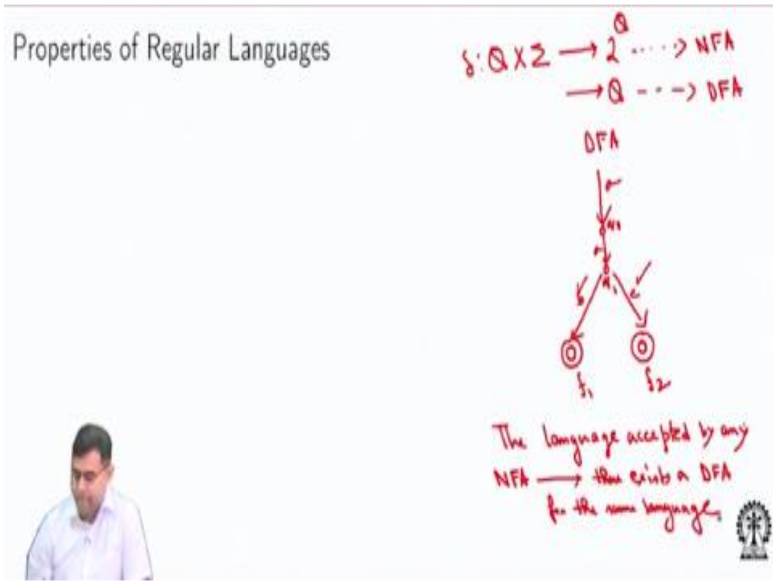
And over this we have the set of context sensitive languages, language is that are accepted by what we call as linear bounded automatons. So, linear bounded automatons are essentially nothing but tuning machines with a finite tape plane. So, we are not going into that detail. This is just for a general picture here. The Turing machines are the highest level of computing power, they represent the highest level of computing power.

That means it is a machine with a head that can read or write on an input tape. The there is some input on the tape and the machine can read and write on the tape based on some transition function and the tape is infinite at least on one side. Now when I make the tape bounded it is so that creates a restriction on the machine and it is known as linear bounded automaton. So, the bound is a function of the input string size in a linear relation.

So, that is my context sensitive languages and they pretty much subsume the set of PDFs which also means that they pretty much subsume the set of regular languages. And at the highest level we have languages accepted by Turing machines. So these are the unrestricted language class. So, this is pretty much your Chomsky hierarchy or name so basically the language hierarchy named after the imminent researcher I mean Chomsky.

Now the question where we will focus on is regular languages and their properties because they represent languages which are related to finite automatons. So, let us get into that.

**(Refer Slide Time: 22:02)**


Properties of Regular Languages
$\delta : Q \times \Sigma \rightarrow 2^Q$ ····> NFA
$\rightarrow Q$ - - -> DFA
DFA
$f_1$
$f_2$
The language accepted by any NFA ⟶ there exists a DFA for the same language.

So, what we will start with is if you remember when we were talking about the transition function, we wrote something like this. You are in a state and there is an input event. So you go to some state. Now you can either go to a unique state or you can go to a set of possible states any one of them. So, this representation corresponds to what we call as deterministic finite automaton and this representation corresponds to what we call as non-deterministic finite automaton.

Why? Because of course in this case there is a unique state where we want to jump and here what we have is a set of possible states and we want to jump into any one of them. So, let us take some example, of a deterministic finite automaton and a non-deterministic finite example automaton. So, just draw a simple example here. This is the accept states. Is this a deterministic finite automaton? Not really. why?

Because if you see here this there is an input. You come here now from this state, we are saying that based on the input A, I can either go here or go here. So, let us just give the state some name let us call them as $q_0$, this $q_1$, this $q_2$, this $f_1$, this is $f_2$. So, this is not really a DFA because of this issue here because you have two possible states, I mean $q_1$ and $q_2$, where you can jump from $q_0$ based on the same input A.

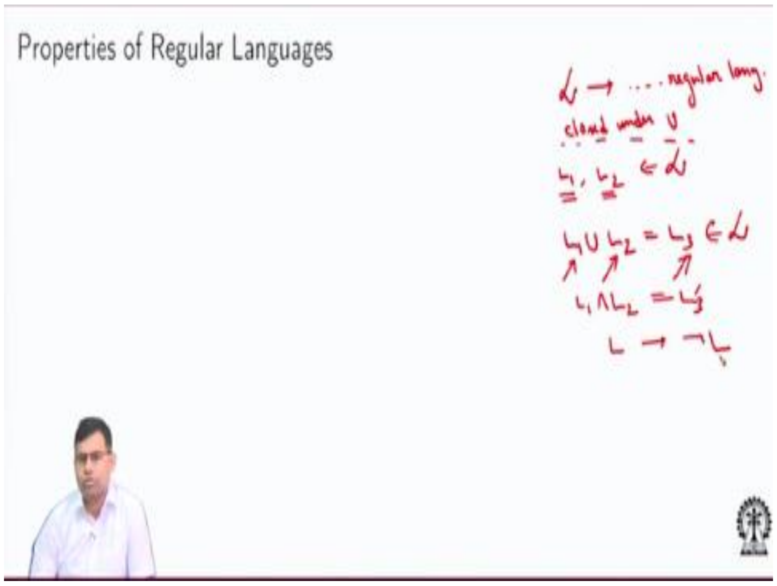
So, what should really be the DFA? Let us see. So, suppose this is the input. You are here, then there is one a. You go here. And then there is there b or c and you either reach here or here. So, look at this automaton. Now can we say this as deterministic automaton, yes, why? Because if you see this automaton, all the transitions are uniquely mapping from a current state through a future state.

From here you go to $q_0$ from initially. Then from $q_0$ with a you go to $q_1$ and then $q_1$ only if it is b, you come here, only if it is c you come to $f_2$. Of course, we does to say that both these and the previous automaton the NFS version they accept the same language. Now there is something important we must understand that while it may seem that NFS have got more modelling power because based on the same input I can go to multiple possible states I mean any one of the multiple possible states.

But still for any or NFA whatever it is its language there will be a corresponding DFA that exists. So, I mean the set of all possible language is accepted by NFS is exactly equal to the set of the language as excited DFS. All that may happen that for any NFA if you are going to create the corresponding DFA its size will be exponential. The number of states that the DFA shall have will be exponential.

So, for the language accepted by an NFA, there exists a DFA for the same language. So, that is more or less of it.

**(Refer Slide Time: 27:44)**



## Properties of Regular Languages

$\mathcal{L} \rightarrow \ldots$ regular lang.

closed under $\cup$

$L_1, L_2 \in \mathcal{L}$

$L_1 \cup L_2 = L_3 \in \mathcal{L}$

$L_1 \cap L_2 = L_3'$

$L \rightarrow \neg L$

Now what we have previously said if you remember that the set of regular languages they are closed under properties like union, intersection and complement standard set theory properties. So, let us first understand what does that statement even mean. So, when I say that this set let us call it this set of all regular languages, I mean that is closed under any operation let us say union, what we mean is pick up two languages from this.

Now languages are also sets. Because they are comprising a set of strings a specific set of strings which will drive and the corresponding automaton from the initial state to the final state. So, when I take these sets and I can operate them with a standard set operator that is union this will give me another language let us say $L_3$. Closed under union means this $L_3$ which is resulting from this union must also be a member of this set of regular languages.

Which means $L_3$ if to put it in another way if $L_1$ and $L_2$ are regular, $L_3$ must also be regular that is all we want to say. Similarly, if $L_1$ and $L_2$ are regular, then $L_3$' which is an intersection there must also be a regular. Similarly, if L is regular then negation of L that means the complement set of the language must also be regular that is what we are saying here. With this we will end this session. Thank you for your attention.