Theory of Computation

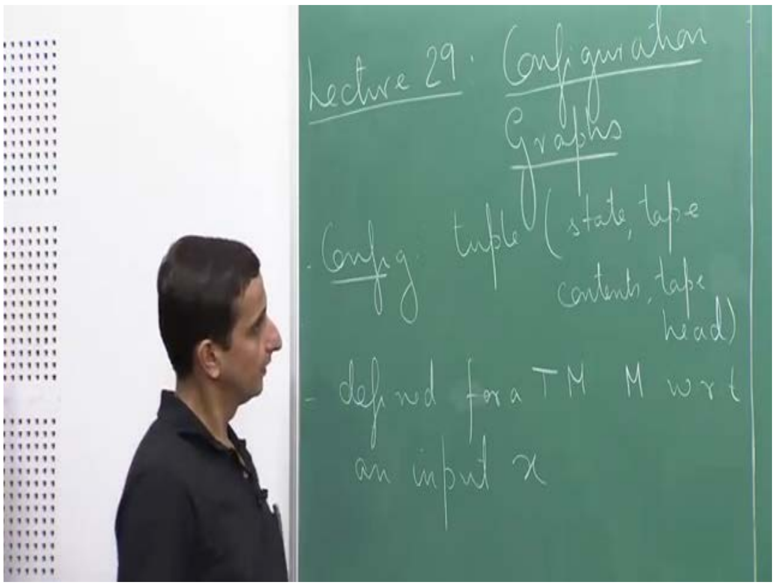
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 29
Configuration Graphs

Welcome to lecture 29 of this course. So, today we are going to introduce the formal definition of a Configuration.

So, we have been using this term in several lectures, but we are going to look at the formal definition today, and using the definition of a configuration we will define what we mean by a configuration graph. Once again, we will see that the concept of a configuration graph will allow us to explain many other properties of Turing machines in a much more efficient and easy to understand manner.

(Refer Slide Time: 01:00)


Lecture 29: Configuration Graphs
- Config: tuple (state, tape contents, tape head)
- defined for a TM M wrt an input x

First let us formally define a configuration. So, we know what a configuration of a Turing machine is so we have seen that. So, just to recall, so a configuration is a tuple of the form state tape contents and position of a tape head. So, this is essentially what is the definition of a configuration. So, a configuration graph, so again once again as it is with the case of a configuration, so a configuration recall that is only define if we fix a Turing machine and an input to the Turing machine.

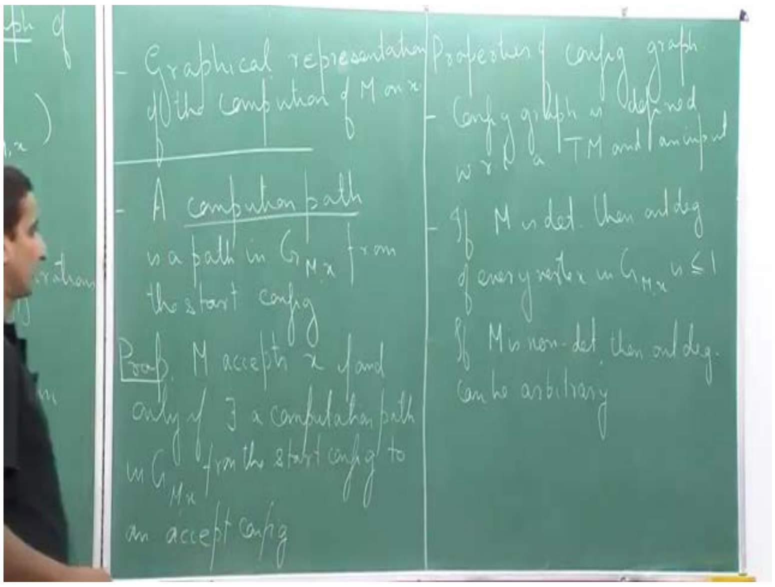
Let us say that we have a Turing machine M, so defined for a Turing machine M with respect to an input say x. So, this is what we mean, because for different inputs again the configurations can be different of course, for different machines the configurations will be different.

(Refer Slide Time: 02:50)

Lecture 29: Configuration Graphs
(state, tape contents, tape head)
TM M w.r.t
x
- A configuration graph of M on input x (denoted as $G_{M,x}$) is a graph whose vertices are the configurations of G w.r.t x
There is an edge from config C_1 to C_2 if the TM M can go from C_1 to C_2 in one step

Now, let us look at the definition of a configuration graph. So, a configuration graph of M, so this graph is denoted as G M x is a graph whose vertices are the configurations of g with respect to x. And there is an edge from configuration c 1 to c 2, if the Turing machine M can go from c 1 to c 2 in one step. So, basically it is essentially a graphical way of modeling the computation of M on an input x.

(Refer Slide Time: 05:02)



- Graphical representation of the computation of M on x

- A computation path is a path in $G_{M,x}$ from the start config

Prop. M accepts x if and only if $\exists$ a computation path in $G_{M,x}$ from the start config to an accept config

Properties of config graph

- Config graph is defined w.r.t. a TM and an input
- If M is det. then out deg of every vertex in $G_{M,x}$ is ≤ 1

If M is non-det. then out deg. can be arbitrary

It is a graphical representation of the computation of M on x. So, basically if you have a transition from one configuration to another in the graph, there is an edge between the first configurations to the second configuration. So, it is a directed graph. Now, in this configuration graph, let us look at this is this definition of a configuration graph. Let us look at one more definitions. So, a computation path is a path in G M x from the start configuration. So, recall once again from last lecture that there is a unique start configuration, so the start configuration is the configuration where the state is q 0, the tape contents is the input that is given that is x in this case and the tape head points to the left most cell of the tape. So, it is a unique configuration.

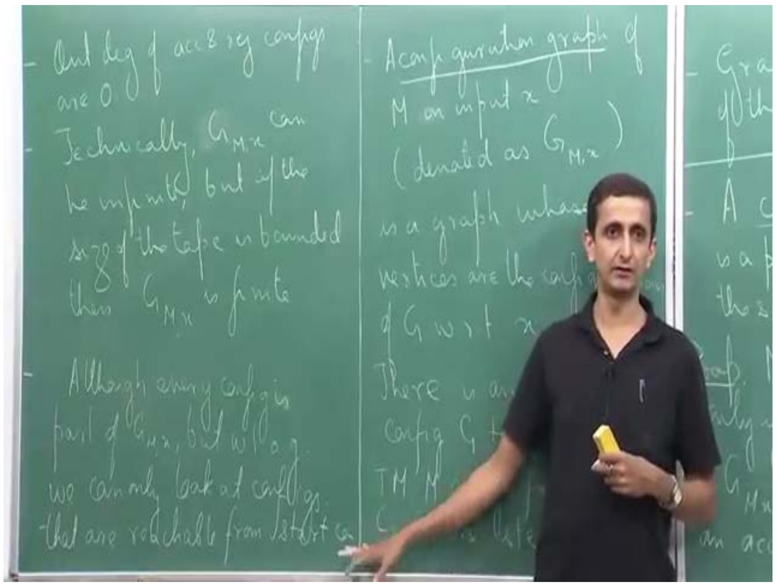
Therefore, there is a unique vertex corresponding to the start configuration in G M x. So, any path in G M x that starts at the start configuration is known as a computation path. So, there can be computation path that goes from the start state to an accept configuration, so that would be a accepting computation path, there can be a path which goes to a reject configuration would be a reject rejecting computation path and so on.

So, therefore, the following proposition is very easy to see M accepts x, if and only f there exist a path or there exist a computation path in G M x from the start configuration to an accepting configuration, because if there is a path from a start configuration to the accepting configuration, I can follow that path and that would accept the input x. If there is no such path that is all paths lead to reject configuration, it means that x is not accepted.

Let us look at some properties of the configuration graph, so properties and notations if you make also. The first property or I should not call it a property it is more like an important observation is that configuration graph is defined with respect to a Turing machine and an input. So, this is just emphasizing what I said earlier that we cannot have a configuration graph of a Turing machine, so that is something which is incorrect. So, if you have a Turing machine, and if we fix and input of the Turing machine only then does it make sense a talk of configuration graph. So, it is a function of two quantities the Turing machine and then input to it. So, given both these two, I can output the configuration graph.

Now let us talk about, let us try to understand how the structure of the computation graphs look like. So, if M is deterministic then out degree of every vertex in G M x is utmost 1 cannot be more than one, because essentially what this means is that if deterministic is that from every configuration there is utmost 1 configuration that you can go to. In the special case, when the configuration is accept or reject configuration, there are no edges going out when which case the out degree is 0; otherwise, it is 1. If m is non-deterministic then out degree can be arbitrary.

(Refer Slide Time: 12:08)



- Out deg of acc & rej configs are 0
- Technically, $G_{M,x}$ can be infinite, but if the size of the tape is bounded then $G_{M,x}$ is finite
- Although every config is part of $G_{M,x}$, but w.l.o.g. we can only look at configs that are reachable from start config

- A configuration graph of M on input x (denoted as $G_{M,x}$) is a graph whose vertices are the configurations of G w.r.t x

There is an
config C_1
TM M
is later

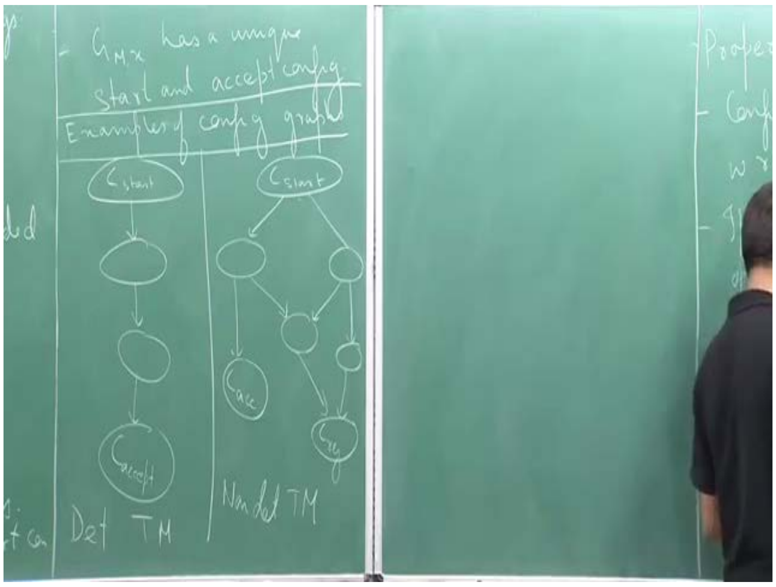
Next is what I just say that so out degree of accept and reject configurations are 0, and this is actually an if an only if, if the out degree is 0 then it is accept; and reject configuration and the other side is what I mentioned. What else, so what about the size of us the graph, so technically. Let try to understand this. So, what would be the size of the configuration graph of a Turing machine on an input, how large can it be? So, technically observe that it can be infinite, because it stores three quantities, it stores the state of course which is finite, but it stores the contents of the tape, potentially you can on a certain input what you can do is that you can just keep on adding symbols to the tape without ever stopping.

Therefore, you just go into an infinite loop that keeps on increasing the tape at every step. The number of configurations can be infinite. So, technically G M x can be infinite. But if the size of the tape is bounded then G M x is finite. Suppose, if you say that you can only access the first hundred cells of the tape or you can only excess if you have an input n, you can only excess the first n square cells of the input of the tape. Then there are only finitely possibilities for those n square many cells and hence the size of G M x is also bounded becomes a finite graph.

And the last point is that all though once again by definition every configuration is part of G M x, but without loss of generality, we can only look at configurations that are reachable from the start configuration. So, all though I mean by definition the graph contains all possible configurations of the Turing machine on that input, but there are certain configurations which are not reachable from the start configuration.

In other words, when you are performing the computation of the machine on the input x you will never even reach that computation. So, what is the point of keep in the graph? So, without loss of generality, we can only focus on the configurations that are reachable from the start configuration. So, this we can always assume without loss of generality.

(Refer Slide Time: 17:12)



Another point that we can assume without loss of generality is that the G M x has a unique start and accept configuration. Because suppose G M x, so technically, of course, G M x can a multiple start and accept configuration, but we can always add a new configuration we can always, for example, in the Turing machine I can add a new state for as the start state which has the following property.

That, if the Turing machine enter the whole start state, what I would do is that I would first clear the entire tape, I will erase the entire tape move the cell to the left most cell of the tape, and move the tape head to the left most cell of the tape, and then enter this new accept state that would be a unique configuration, because the tape contents are empty and the head is pointing to the left most cell.

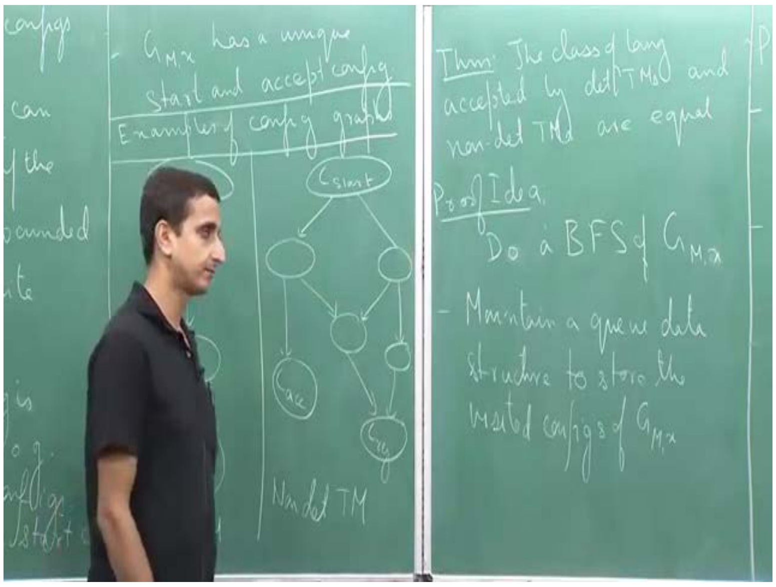
Similarly, if it enters the reject state I can do the same thing; I can empty the contents, I can move the tape head to the left most cell and then I enter the new reject state. Essentially, I can have unique without loss of generality I can assume that G M x has a unique start and accept configuration. So, these are some properties that are I mean that will be used when we talk of configuration graph of a Turing machine. Let us look at so we talked about deterministic and non-deterministic Turing machines, but let us just try to see an example of how their configuration graphs differ. So, here is what the configuration graph of a deterministic Turing machine will look like. So, I have the start configuration. So, I will call it C start. Let us call it C start.

And by its nature, the out degree at every vertex is utmost 1. So, from C start, I can go to a unique configuration; and from here, I go another; and then from here may be I go to a accept configuration let us say C accept, so therefore, it is one. O f course, I can have a loop also I can may be come back to a previously visited configuration that is also fine, but I cannot have more than one out going edge. So, this is so let me write it here examples of configuration graph. So, this is for a determinism Turing machine.

In the case of a nondeterministic Turing machine, I have C start. Then may be from here, I go to two different configurations; and then I come here, and then I go it some configuration over here, here. May be this is C accept and this is C reject. So, there a many computation parts; one that goes from start to C accept; there is another going form C start to C reject; there is another going from C start to C reject; there is a third computation part that is also going from c start to c reject. So, this is the configuration graph of a nondeterministic Turing machine.

In this case, do we accept the input x or not, what do we do. So, observe that, in this case, we should accept the input because by definition there is a computation part that goes to an accept configuration that is the definition of acceptance of a nondeterministic machine. If there is at least one computation path goes to accept configuration, we accept it; and we do not care and if we would have rejected x only if all computational paths had gone to reject configuration.

(Refer Slide Time: 23:03)

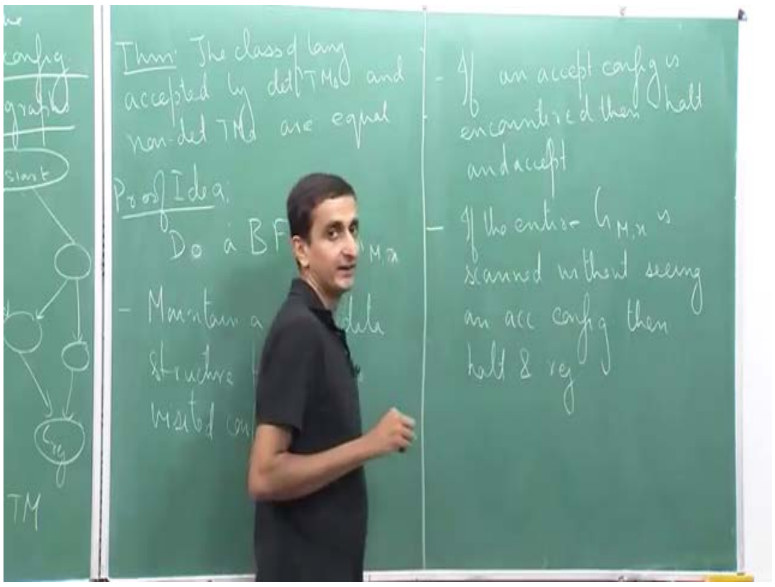

- $G_{M,x}$ has a unique start and accept config
Examples of config graph
C_{start}
C_{acc}
Non det TM
Thm: The class of lang accepted by det TMs and non-det TMs are equal
Proof Idea.
Do a BFS of $G_{M,x}$
- Maintain a queue data structure to store the visited configs of $G_{M,x}$

Let us look a quick application of this idea. So, we will state this as theorem. The class of languages accepted by deterministic Turing machines and nondeterministic Turing machines are the same. So of course, by definition, nondeterministic Turing machines are more powerful though than deterministic Turing machines, or assume some more, but at least as powerful as deterministic Turing machine, but what this theorem says is that they are not any more powerful. So, there equal in parts. So, whatever we can do with a nondeterministic Turing machine, we can do with a deterministic machine as well.

So, how do we prove this? The proof idea is if I want to just state it one line it is to do B F S of G M x. So, suppose we are given a Turing machine and nondeterministic Turing machine M and I am given an input x, and I want to decide whether x is accepted by M or not, which is non-deterministic. What I do is I compute the computation of tree of G M x and I actually do not compute it all together I just compute it as when required.

First I compute the start configuration, which is of course, easy because it is just the input the start state and input head is at the left most cell. Now I look at the transition function of the non-deterministic Turing machine, and I keep on computing what are the children of the currents state from each state. So, maintain q data structure to store the visited configurations of G M coma x.

(Refer Slide Time: 26:24)


config graph
start
C_{rej}
TM
Thm: The class of lang accepted by det TMs and non-det TMs are equal
Proof Idea:
Do a BF
$_{M,x}$
- Maintain a
data
structure
visited con
- If an accept config is encountered then halt and accept
- If the entire $G_{M,x}$ is scanned without seeing an acc config then halt & rej

If an accept configuration is encountered then halt and accept; and if the entire G M x is scanned without seeing an accept configuration then halt and reject, so that is the idea. So, basically what we do is we scan the computation graph level wise, first level, second level then this is at the third level and so on. If we ever see a accept computation, we know that there is a path from the start configuration to the accept configuration, and we halt and accept.

And if we exhaust the entire graph without ever visiting the start configuration we then halt and reject, because we know that there is no path from the start to the accept configuration. But what can also happen is that maybe it is an infinite graph, maybe there is one computation path with never ends that can very well happen. In which case your deterministic simulation would also go forever, it is a deterministic simulation which would also gone forever which is still fine. But the point is that if there is an accept configuration, it will be at some level and that level will always be encountered.

So, I will stop here, and we will continue next time again.