Model Checking
Prof. B. Srivathsan
Department: Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture - 26
Simple properties of Bchi Automata

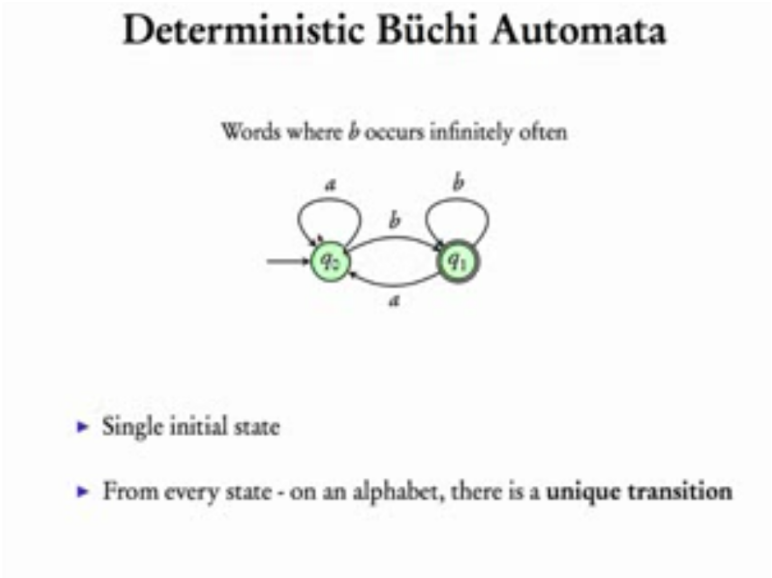
In the last module we saw this concept of non deterministic Buchi automata. We saw how these automata can be used to accept certain languages over infinite words and NBA looks like an NFA however the accepting condition says that a word if accepted if it has a run where in accepting state is seen infinitely often. That is the accepting criterion for an infinite word.

(Refer Slide Time: 00:40)

Similar to the case of NFA, we will now see some simple properties of this non deterministic Buchi automata. In the previous module we have seen a couple of examples. If you are clear with those examples, you can proceed with this module. In this module we will see more constructions of Buchi automata and this entire concept of Buchi automata would become more and more clear. Let us first look at deterministic Buchi automata.

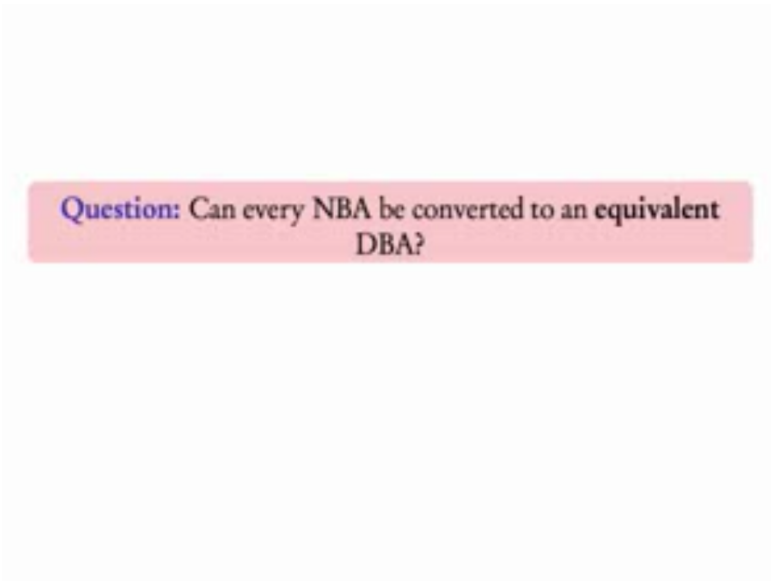
This is the part we are going to look at determinization of non deterministic Buchi automata.

(Refer Slide Time: 01:26)



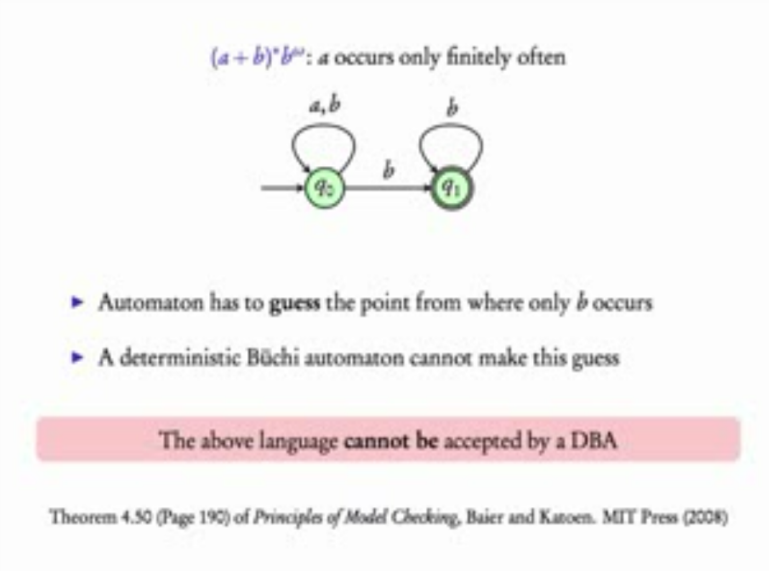
When this Buchi automata said to be deterministic, it should have a single initial state as usual and from even state on an alphabet there should be an unique transition. This is the same as in the case of a DFA. So you cannot allow a from the state for example. Now this deterministic Buchi automaton accepts all words where b occurs infinitely often. We have seen this before.

(Refer Slide Time: 01:59)



Question, can every NBA be converted to an equivalent DBA? Remember that every NFA could be converted to a DFA which accepts the same language and how do we did that using what is called the subset construction. So we ask a similar question, can we do something like subset construction in the case of Buchi automata. Can every non deterministic Buchi automaton be converted by the equivalent deterministic Buchi automaton, is this possible?

(Refer Slide Time: 02:39)



Let us look at a language. It says that a occurs only finitely often, a plus b star b power omega. Here is the NBA corresponding to this and why this is not deterministic? It not deterministically chooses the point from where it can see only b. If you give a run, if you give a word in this language, at some point it jumps to seeing only b and the automaton at that point will jump to q1.

Here is no determinization because from q0 you can either b or you can take this transition or this transition. This transition is when you expect to see some more is and you take this transition when you know that you will not be able to see anymore b. This is what is written. If you take word in this language, there will be one run which correctly guesses the point from where only b occurs and from there you will see only q1.

It turns out that non determination is crucial for accepting this language. A deterministic Buchi automaton cannot make the guess that from now on, I will see only b. This kind of a guess cannot be made by deterministic Buchi automaton. It turns out that this language cannot be accepted by any deterministic Buchi automaton in the sense that you cannot construct a DBA for this language.

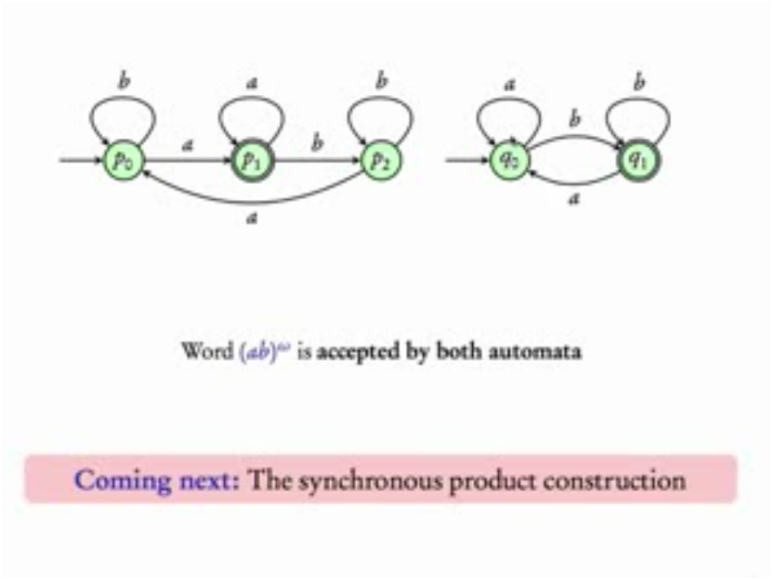
I have given you just the intuition, a proper proves of this statement can be found in the book in page 190. So it turns out that non deterministic Buchi automata have more power, than deterministic Buchi automata in the sense that there are languages that non deterministic Buchi automata can describe but deterministic Buchi automata do not have the capacity to describe such languages.

(Refer Slide Time: 05:07)

Here is the summary. DBA are less powerful than NBA. In the finite case, in the case of finite words this was not true. Non determination did not add any special power to the automaton, any deterministic automaton NFA could be converted to an equivalent DFA. So you can see the language in a deterministic way. However, in the words of infinite words, NBA are more powerful that DBA. This is something we need to know.

This is the first change in properties as compared to NFA. Let us now look at the product construction and see what happens.

(Refer Slide Time: 05:57)



Consider these two automata, what is this automaton? On an a it goes to its accepting state. As long as it sees a b it keeps lopping here. On an a it comes here and as long as it sees a b it

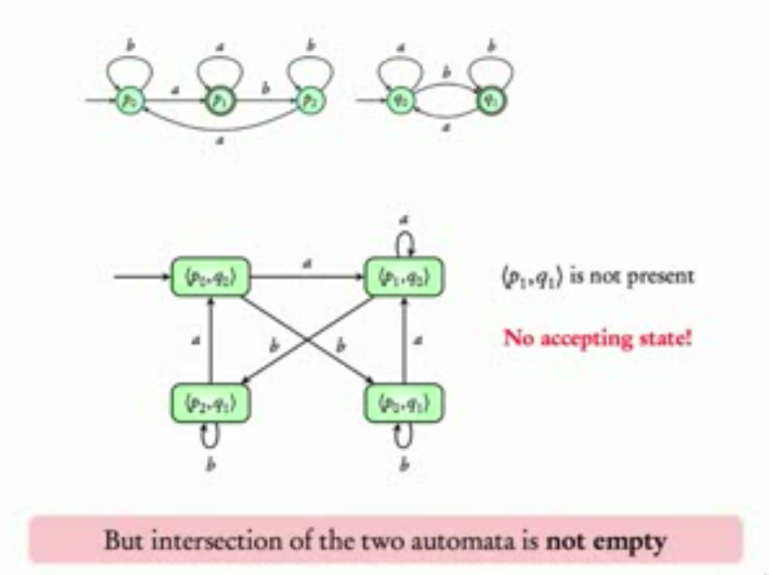
is easier. Again on a it comes here and when it sees an a it keeps looping here. So this automaton accepts all words where a occurs infinitely often. For example, a b a b a b. a b and so on. It sees p1 infinitely often. Now what about this automaton.

This is an automaton which sees D infinitely often. Note that for that language we could have given an automaton with two states but for an illustration I have given this automaton. This bigger automaton also accepts the set of words where a occurs infinitely often. What is common to these two? The word a b power omega is one word which is in the intersection of these two.

So take these automaton as well as this automaton, a b omega, a b a b a b so on, q1 occurs infinitely often in this run and it is accepting. So a b omega is accepted by this automaton and it is accepted by this automaton as well. Now let us try to form this synchronous product of these two automata. Recall that in the case of NFAs, we did this synchronous product to get the automaton for the intersection of the two languages.

Let us now see, if we can see the same synchronous product to get the intersection of these two languages.

(Refer Slide Time: 08:03)



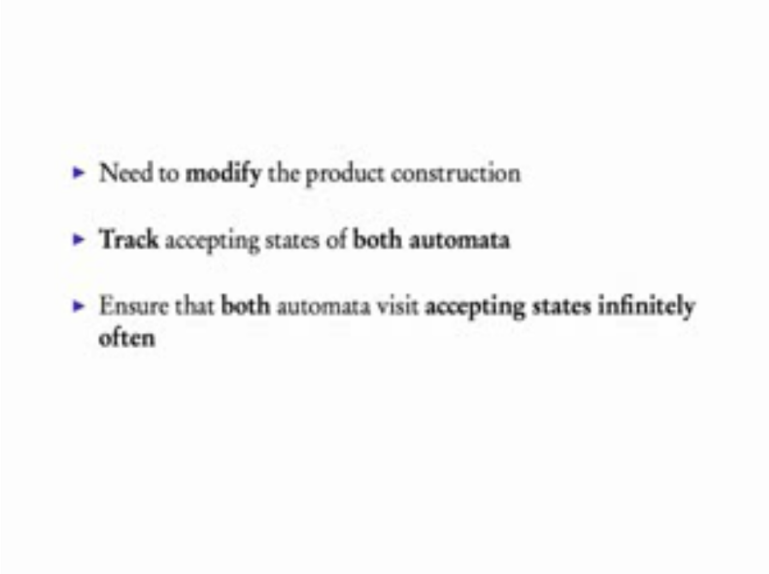
Here is the synchronous product. You start at p0 q0 on an a, p0 goes to p1, q0 stays at q0, so you do this. And on a b p0 stays at p0 and q0 goes to q1, so you have this transition. Similarly, this is the standard construction that I explained in the last unit. From p1 q0, p1 on

an a, stays in p1 and q1 on an a stays at 0, so you have this loop. Similarly on a b, p1 goes to p2 and q0 goes to q1. So you have this transition.

If you have a look at it you will be able to recall the synchronous product that we saw in the last unit. What about the accepting stage, if you remember, we made some state accepting only if both the states here were accepting. Here p1 is accepting and q1 is accepting. There does not seem to be a state containing p1 and q1. So p1 q1 is not present. That means that in the synchronous product there is no accepting state which means that the language of the synchronous product is empty.

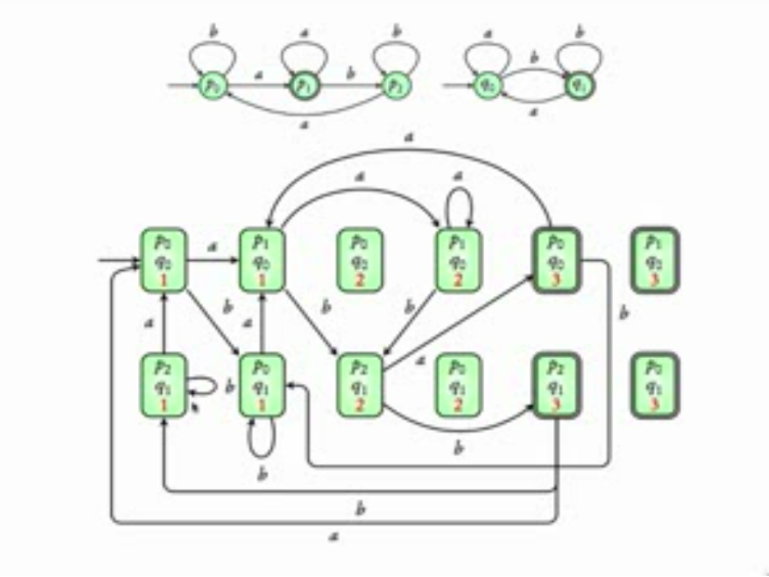
But we know that the intersection of these two automata is not the empty language. There is at least one word. So this kind of simple synchronous products will not give the intersection. What we should be doing is that, we should try to take a product and we should track the accepting states of both the automata. We should be able to say that on this word both the automata would have visited accepting states infinitely often.

(Refer Slide Time: 10:14)



This is what I have explained here. We need to modify a product construction to track the accepting states of both the automata and somehow we need to ensure that both the automata visit accepting states infinitely often.

(Refer Slide Time: 10:34)



Let us slowly take a look at this construction. We have automaton 1, automaton 2, what would the states be in the product? So what I have done here is that, in addition to p0 q0, we also have a flag and this flag can take values 1,2 and 3. If you look at it, here we had four states. Similar to that, we will have 4 states with one, 4 states with two and 4 states with three. And the states with 3 will be marked accepting.

Now what is the role of this flag. The states with flag one are waiting to see an accepting state of the first component, the states with flag 2 are waiting to see with an accepting state of the second component. In fact they would have already seen an accepting state of the first component and now they are waiting to see an accepting state of the second component. The ones with flag 3 tell you that accepting states of both the components have been seen.

Well if this intuition is not clear, let me continue with the construction and then you will see how this works. Look at p0 q0. What happens on an a? p0 goes to p1, q0 stays in q1. Is p0 an accepting state? No. So what you do is, on an a, you go to p1 q0, don't change the flag. Look at b, p0 on a b stays in b, q0 on a b goes to q1. So this part is clear. You have to go to p0 q1, however should we go to the p0 q1 2 or the p0 q1 3. For doing that you look at p0.

If this is one you look at p0. Is p0 an accepting state, no. So you stick to 1. Let us look at this transition, p1 on a, stays in p1, q0 on an a stays in q0. So in principle we would have given a transition from p1 q0 to p1 q0 however what about the one. This one as I said is waiting to see an accepting state of the first component. Since p1is an accepting state you should move to the state with 2.

This means that yes, we have seen an accepting state of the first component. So let us now move to a stage where we are waiting to see an accepting state of the second component. So p1 q0 on an a, goes to p1 q0 since p1 was accepting. One will be incremental and we go to 2. What about this, say p1 on a b goes to p2, q0 on a b goes to q1. So p1 q0 should go to p2, q1. And since p1 is an accepting state, this one is increased and you move to 2.

Let us have a look at this state, p0, q1, p0 on an a goes to p1, q1 on an a goes to q0, is p0 an accepting state no. So one should be changed. This is what happens, this is similar to the previous case where p0 q1 on an a goes to p1 q0. And p0 q1 on a b stays here, since p0 is not in accepting state you do not change the one. let us now look at this state. In this state we are waiting to see an accepting state of the second component. So the procedure is simple.

You look at p1 q0, p1 q0 on an a should go to p1 q0, however there are 3 possible p1 q0. You look at the flag, see if q0 is an accepting state because this is two, you should look at this. Is q0 is accepting state, no. So we should stay here itself. You should go the p1 q0 with a 2. Similarly, p1, q0 on a b should go to p2, q1 and since this q0 is not accepting state, it goes to the b2 q1 with the same flag. Let me try to look at transitions out of this state.

P2 on an a goes to p0, q1 on an a goes to q0, moreover q1 is an accepting state. So p2 q1 should go to p0, q0 3 on an a. This is what happens. Similarly p2 q1 on a b stays in p2 q1, but since q1 is in accepting state you move to 3. If you see, look at the path like this, this would tell you that if you have come to 3, you have seen an accepting state of 1 and an accepting state of automaton 2.

Look at this path, a, b, a, in this, so this path is mimicking the runs of both automata. The top component talks about the first automaton and the second component talks about the runs of the second automaton. So look at the run, p0, p1, p2, p0. So we have seen an accepting state of one. Look at the second component, q0 q0, q1 qo and q1 is an accepting state. So we have seen an accepting state of the second component as well.

Similarly look at the path that comes here, there are many ways of doing it., this any number of times do this and come here. If you see all paths that come here will have seen an accepting state of the automaton 1 and an accepting state of the automaton 2. For example, p0

p1, p1 p2, p1 is in the accepting state that we have seen and here q0 q0, q1 q1. So q1 is an accepting state.

Essentially we are trying to track the runs of both the automata by doing this product and additionally this flag is there to track the fact that both the accepting state of 1, as well as the accepting state of 2, have been seen. And for doing that, that is why whenever the top component is in accepting state and we are here, we move to a state where one becomes two. Similarly, when we are in two and the second component is in accepting state, we moved to 3.

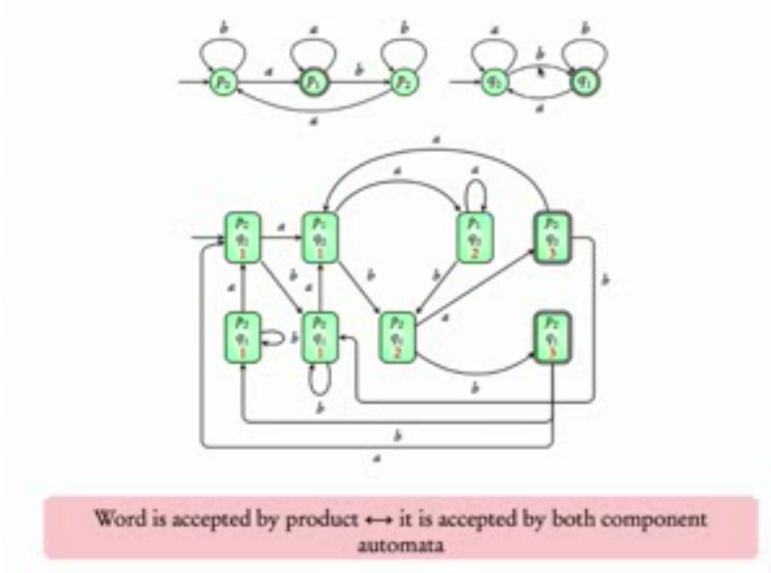
At 3 you always get back to 1 so that this can be cycled. Look at p0 q0, p0 q0 on an a should go to p1 q0. It should go to p1, q0 1 on an a. So this is what happens. On a b p0 should stay in p0 q1. So p0 q1 1 have come back here. So now we want to see the same thing again. So in the sense that we have come to a state where we are waiting to see an accepting state of one. Remember that we want to track the fact that the accepting state of 1 occurs infinitely often and the accepting state of 2 occurs infinitely often.

They can occur in different part of the run but both of them should occur infinitely often. So that is why once we know that, now we have seen both. When we go back and if we come back here again we will know that we have seen it twice and if we come back here again, we will know that we have seen it thrice, four times and so on. Since this is the accepting state if there is a run where this state occurs infinitely often or this state occurs infinitely often.

It would mean that we have seen the accepting states of both, this automaton as well as this automaton. Let us finish this construction and I will illustrate this idea or more words. We are here, p2 q1, p2 on an a goes to p0, q1 on an a goes to q0. Since we are at 3, we should go to p0, q0 1. So on an a, you go here. Similarly, on a b we go to p2 q1 1. Now we are left with this state, p2 q1 on a 1.

So p2 on an a goes to p0, q1 on an a goes to q0. Since p2 is not an accepting state you should not change the flag. Hence on an a you go here, on a b you stay here.

(Refer Slide Time: 21:34)



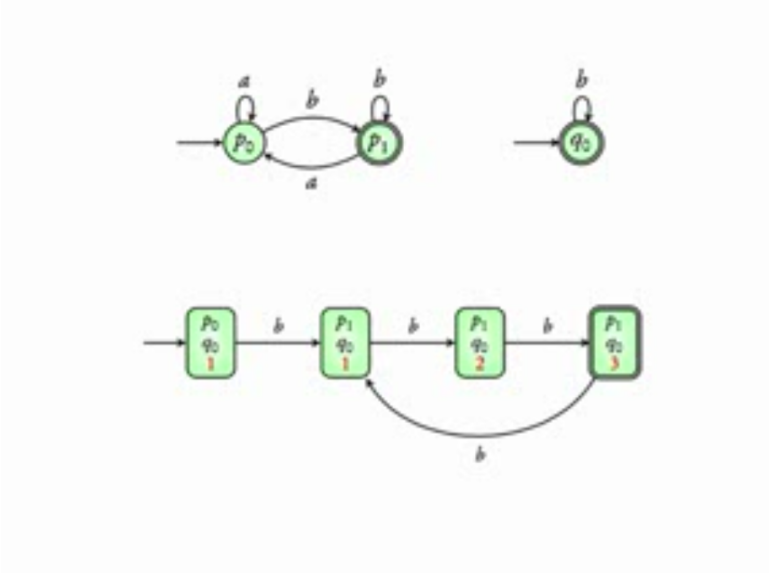
Let us remove the unreachable states, you will now get an automaton. These are the accepting states. Look at this as a non deterministic Buchi automaton. Well, in this case it is even deterministic thing. So let us look at a b a b a b a b. So this thing will keep cycling and you will see that this accepting state would be seen infinitely often.

Essentially if you work this out on examples, you will figure out that this automaton is tracking the runs of both these automata and finding whether both of them have an accepting run on that word. For example, here when we did a b a b, we were doing p0, p1, to p2 and so on and that is what is happening here, p0 to p1 to p2 to p0 back to p0, p2, p0, p0 and so on. So this is the run, a b a b a b and so on.

And the second component would be tracking the run of a b a b a b on this automaton. So for any word, any infinite word this product automaton with this flags would be tracking the runs of both these automata and also would be tracking the fact that both of them, whether both of them are seeing their accepting states infinitely often. So here we are, a word is accepted by this automotive if and only if they are accepted by both the component automata.

So this automaton would give you the Buchi automaton corresponding to the intersection of these two languages.

(Refer Slide Time: 24:10)



Let us look at one more example. This is a Buchi automaton for the language consisting of infinitely many of these and this is the Buchi automaton for the language consisting of b power omega, the singleton b power omega. So the intersection of these two should be what? Should be just b power omega. Let us try to get this through our synchronous product. Here is the automaton corresponding to the synchronous product. So we start at p0 q0 with 1.

You read only a b, because q0 can see only a b and when we read a b we go to p1 q0. Since p0 is not an accepting state the flag doesn't change. Now from p1 q0 on reading a b, you stay in p1 q0, however since p1 is an accepting state we have already seen an accepting state of the first component, so you should to 2. Now p1 q0 on a b goes to p1 q0. Since q0 is an accepting state, we have seen an accepting state of second component.

So we go to three and here again on a b you go to p1 q0 start from 1. Note that we just have to track the fact that the accepting states occur infinitely often. We are not obliged to track every occurrence of the accepting state. For example, here q0 was accepting for the second component. However, we were interested only to look at the first component when we were here.

Similarly, here p1 was an accepting state but when we were here we are interested only in checking if 2 was in its accepting state. As long as we loop around this, we will know that both this and this automaton have been visiting in accepting states infinitely often.

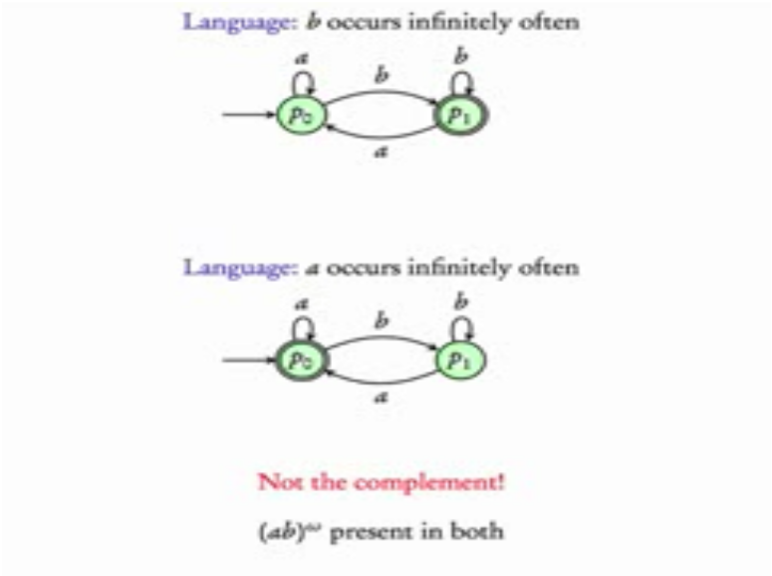
(Refer Slide Time: 26:19)

So we have seen two things. The first thing is that, in this model of automata for infinite words, deterministic Buchi automata are less powerful than non deterministic Buchi automata. Secondly we have seen this product construction with this additional flag and this will give us a Buchi automaton which accepts the intersection of the component automata. The third important thing is that, given a Buchi automata can we say whether the language by this Buchi automata is empty or not.

In the case of NSA, we had to do a depth-first search to check if the accepting state is reachable. But now we need to check if this accepting state can be visited infinitely often. So is there an execution where an accepting state can be repeated infinitely often. This is different from the normal depth-first search. We will see an algorithm for checking emptiness in the next unit.

In this module we will end with the concepts of complementation and Union for languages over infinite words.

(Refer Slide Time: 27:50)



Look at this language. This is the automaton for b occurs infinitely often. This is deterministic as well. Recall that in the case of DFA, we interchanged the accepting and non accepting state to get the complement language. Let us try to do that. What is that we get when we interchange the accepting and non accepting state? We will get the language where a occurs infinitely often. But note that this language is not the complement of this.

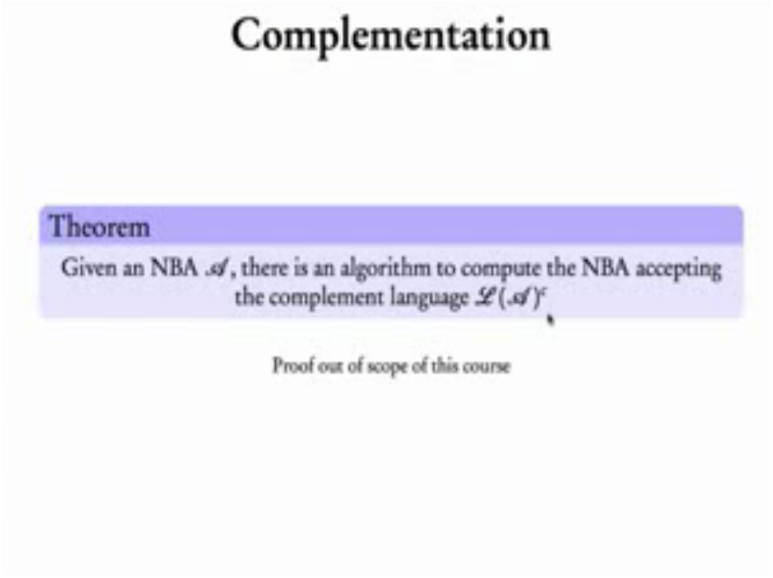
For example, a b omega is present in both of them. In a b omega both a and b occur infinitely often. The complement of b occurs infinitely often is the language b occurs finitely often. Here in this language it just says that a occurs infinitely often. It does not say whether b occurs finitely or infinitely. So doing this mere interchanging of accepting and non accepting states, will not give you the complement language in general.

(Refer Slide Time: 29:17)

So this is a big challenge. How do we now find the automaton for the complement language? Can we even find one? And we need to accept the fact that doing a mere interchange does not work and more over non deterministic Buchi automata are more expressive than deterministic Buchi automata. Recall that in the case of finite word languages, given an NFA, we first converted into DFA and then did this interchange.

In this case first of all we cannot convert, every NBA to a DBA. Moreover, doing this interchanging of accepting state also does not work. So how do we try to get the automaton for the complement.

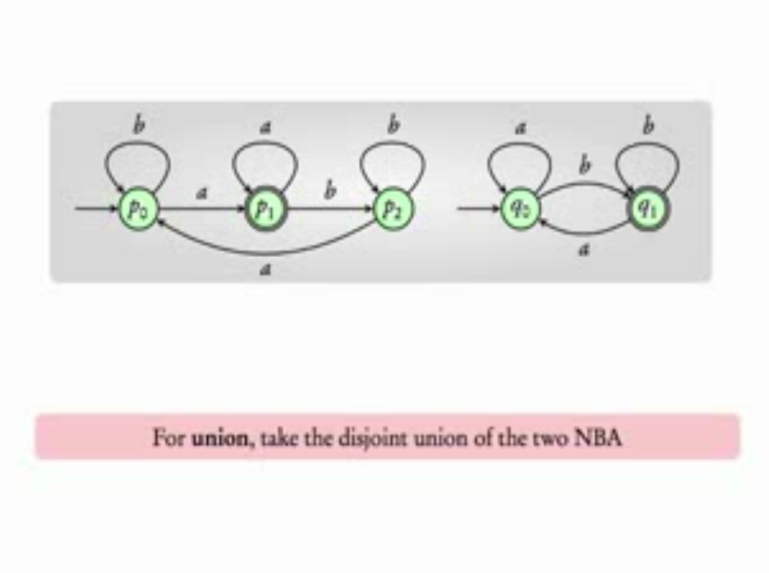
(Refer Slide Time: 30:11)



It turns out that we can indeed compute the NBA accepting the complement language. However the proof of this and the method of doing this is out of scope of this course if you search for Buchi automata complementation you will get a lot of references. This is for the more interested ones who would like to have a look at the proof of this theorem. It is a very non-trivial theorem.

And there is a direction of research to find better ways of complementing NBA and it is turning out to be a tough task. For our course it is enough if we know that given an NBA A there is an algorithm to compute the NBA accepting the complement language.

Refer Slide Time: 31:12)



Finally, we will now look at unions. Suppose if I give you these two automata. These two are Buchi automata and we want to represent the union of these two languages. It is simple, we do the same thing as we did in the case of NSA. We look at this entire thing as one automaton. For any word, the automaton can non deterministically choose to start reading that word either from this automaton or from this automaton because there are multiple initial states. So here it is similar to the finite case.

(Refer Slide Time: 32:01)

We have seen 3 important concepts, determinization, product, complementation union, things are different from the finite case. DBA are less powerful than NBA and for doing this product construction we had to add extra flags, complementation was not very trivial and we have not see the exact proof of complementation. Union was only one where we just with the same thing as before. So this week we have seen ways of handling infinite word languages.

What we will do next is that, we will give an algorithm for emptiness of Buchi automata and what this would lead to is that given a transition system and a Buchi automaton we can check if all traces of the transition system are contained in the language of this Buchi automaton. We will also see that omega regular expressions and all the GF etc. all these can be looked at as Buchi Automata. So this Buchi Automata are a fundamental concept.

We need to know that the emptiness of this Buchi Automata can be done and this product construction would give us the language intersection and the complementation and union can be done for this automaton.