

Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 12
Proving Equivalence of Regular
Expression and DFA Through a GNFA

(Refer Slide Time: 00:17)

Equivalence of reg. expressions and regular languages

Theorem 1.54: A language is regular iff some regular expression describes it. if and only if

This is yet another characteristic for regular languages. We prove it in two parts:

(1) IF (\Leftarrow) and (2) ONLY IF (\Rightarrow)

Lemma 1.55: If a regular expression describes a language, then it is regular.

In this lecture, we will prove the other direction.

In this lecture, we will prove the other direction.

Lemma 1.60: If a language is regular, then it is described by a regular expression.

Proof Outline:

Regular language \Rightarrow Recognized by DFA

\Rightarrow Recognized by GNFA

\Rightarrow Described by a reg exp.

GNFA: NFA that has regular expressions in its transitions, not just symbols in Σ or ϵ .



GNFA: NFA that has regular expressions in its transitions, not just symbols in Σ or ϵ .

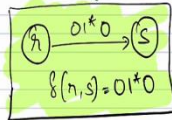


Def 1.64: A generalized nondeterministic finite automaton (GNFA) is a 5-tuple



$(Q, \Sigma, S, q_{start}, q_{accept})$, where

$$\delta: (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R$$



Set of all reg exp over Σ .

If $\delta(q_i, q_j) = R$, where R is some reg exp,



If $\delta(q_i, q_j) = R$, where R is some reg exp,

this means that the GNFA can transition from q_i to q_j when it reads some $w \in R$.



A GNFA accepts $w \in \Sigma^*$ if w can be written as $w_1 w_2 \dots w_k$ where $w_i \in \Sigma^*$ and there exists a sequence of states q_0, q_1, \dots, q_k such that

- (1) $q_0 = q_{start}$
- (2) $q_k = q_{accept}$
- (3) For each $1 \leq i \leq k$, $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$



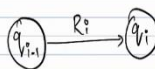
if $\delta(q_i, q_j) = R$, where R is some reg exp,

this means that the GNFA can transition from q_i to q_j when it reads some $w \in R$.

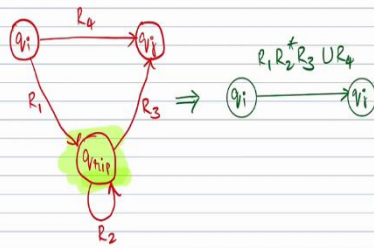


A GNFA accepts $w \in \Sigma^*$ if w can be written as $w_1 w_2 \dots w_k$ where $w_i \in \Sigma^*$ and there exists a sequence of states q_0, q_1, \dots, q_k such that

- (1) $q_0 = q_{start}$
- (2) $q_k = q_{accept}$
- (3) For each $1 \leq i \leq k$, $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$



The key idea of the proof is in reducing the number of states of the DFA by one.



Lemma (b): Returns R & recursively calls itself

1. $k = \text{no. of states of } G$

Lemma (b): Returns R & recursively calls itself

1. $k = \text{no. of states of } G$
2. If $k=2$, then return R.
3. If $k > 2$, then select a state q_{nip} such that $q_{nip} \in Q \setminus \{q_{start}, q_{accept}\}$.

For all pairs q_i, q_j which is not q_{nip} , do the above transformation.

$$G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$$

$$Q' = Q \setminus \{q_{nip}\}$$

$$\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$$

2. If $k=2$, then return R.

3. If $k > 2$, then select a state q_{nip} such that $q_{nip} \in Q \setminus \{q_{start}, q_{accept}\}$.

For all pairs q_i, q_j which is not q_{nip} , do the above transformation.

$$G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$$

$$Q' = Q \setminus \{q_{nip}\} = Q - \{q_{nip}\}$$

$$\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$$

4. Call Lemma (b).



Now, the next part. So, this was the first part of the proof, which is, I am not going into details, but it is fairly clear. The second part is how we go from a k state GNFA to a $k-1$ state GNFA, how do we reduce one state of the GNFA. This is what I want to describe next. This, I will do it in a bit of detail. So, this is the key or main idea of the proof where we reduce the number of states in GNFA by 1.

Basically, what we do is we identify one state that we want to remove from the GNFA. Let us say it is this highlighted state. I am calling it q_{rip} . So, I wanted to slip that off, rip off that straight from the GNFA. So, now how does it impact the GNFA? So, basically you can, earlier, we can go from q_i to q_j by seeing a symbol or a string from R_4 .

You could also go from q_i to q_j by seeing a string that is from R_1 concatenation R_3 . R_1 concatenation R_3 means a string from R_1 will take it to q_{rip} , and a string from R_3 will take you to q_j . So, you could go from q_i to q_j through $R_1 R_3$ also. You could also go from q_i to q_j by processing a string $R_1 R_2 R_3$ because you could go from q_i to q_{rip} . R_2 will keep it at q_{rip} and then R_3 will take it to q_j . So, you could also go from q_i to q_j by seeing a string from $R_1 R_2 R_3$, concatenation of three regular expressions.

It could even be $R_1 R_2 R_2 R_3$. So, the first string from R_2 keeps it at q_{rip} , the second string from R_2 also keeps it to q_{rip} , and then you go from q_{rip} to q_j . So, it could be multiple times. So, we may not use any instance of R_2 , we may use one instance, two instances or many more instances. So, the key thing to note here is that, in summary any string that is from this particular regular expression, so basically $R_1 R_2^* R_3$ will take you from q_i to q_j through q_{rip} .

And now that q_{rip} is removed, we need to, we need to preserve this information. So, we will include that in the arrow from q_i to q_j . So, in the arrow from q_i to q_j we want to add this information that $R_1 R_2^* R_3$ but then whatever was already there we are not going to lose that. So, we want to keep, retain that as well. So, what was earlier there, R_4 was an existing label of q_i to q_j , that we will preserve.

So, now any string from $R_1 R_2^* R_3 \cup R_4$ will take you from q_i to q_j in the old, in the red GNFA. But now in the green GNFA, we do not have the state q_{rip} . So, we will include that in the label from q_i to q_j . So, this is the main idea. So, this basically indicates if a state is removed, so the removed state being q_{rip} , what should be the change in the labels of the other states. So, how to relabel the other states if a certain state is removed.

So, this is how you re-label. So, basically if you want to remove q_{rip} you look at, for any pair $q_i q_j$, you see what is, what is the arrow from q_i to q_{rip} , q_{rip} to itself, q_{rip} to q_j and then notice what is R_1, R_2, R_3, R_4 et cetera, and then use that to label the new arrow from q_i to q_j . We need to do that for all the pairs $q_i q_j$, in both directions. So, if, for $q_i q_j$, we identified the labels, for $q_j q_i$ also, we will have to identify the appropriate labels.

And even in the states the q_i to itself also, we will have to identify the appropriate labels because from q_i there would have been a self loop but then you could go from q_i to q_{rip} and then q_{rip} to q_i . So, in general, let me just say q_i to q_j . This yields the following procedure. It is a recursive procedure, which talks about how to convert G. Let us say G is a GNFA and we want to convert G from k states to k-1 states.

And we want to keep doing that till we get to two states. So, if k is 2 then we know there is only one transition and that the label of the transition will be the regular expression that is desired. If k is more than 2, what we will do is reduce 1 state. So, what do we do? Then we identify a state called q_{rip} , it could be any state except for the start or accept and what we do is for all the pairs of, all the other pairs q_i, q_j , we do the transformation that we just described here. So, basically you remove q_{rip} , you relabel the arrow q_i to q_j with $R_1 R_2^* R_3 \cup R_4$.

$$G' = (Q', \Sigma, \delta', q_{start}, q_{accept})$$

$$Q' = Q \setminus \{q_{rip}\}$$

$$\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$$

So, what we do here, this is just described very briefly. We move from G to G'. So, G' is a new GNFA. It operates on the same alphabet sigma. It contains the same starting and accepting state as G. Two things change. One is the set of states. From Q you remove q_{rip} . So, $Q' = Q \setminus \{q_{rip}\}$, this is set difference. So, this is the same as Q minus q_{rip} . So, it is all the states that were in G except q_{rip} . So, this is this backslash is another way to denote said difference.

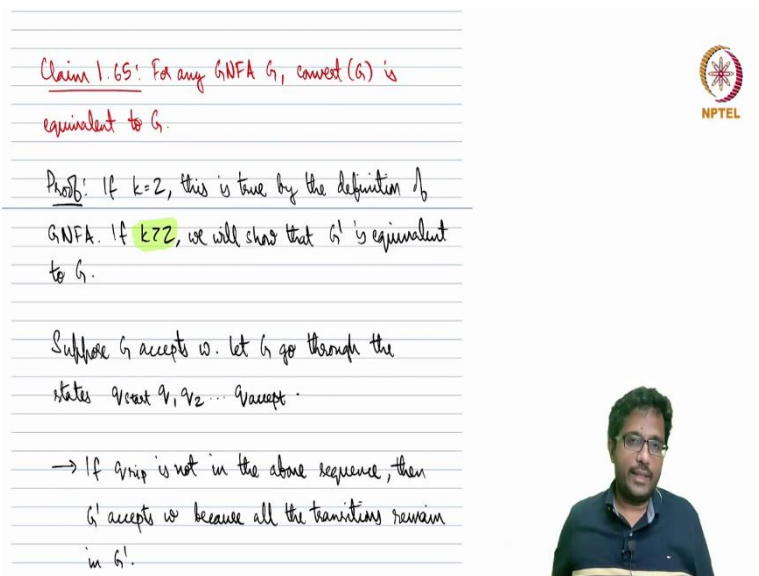
And the other thing is delta prime from the transition function changes. So, if for any q_i, q_j which are in the new machine, as long as q_i and q_j are not the q_{rip} , we identify what are R_1, R_2, R_3, R_4 based on the connection with q_{rip} and then the, we relabel it with $R_1 R_2^* R_3 \cup R_4$.

So, the earlier label would have been R_4 . So, now we add using union $R_1R_2^*R_3$ where R_1 is the label from q_i to q_{rip} , R_2 is the label from q_{rip} to itself and R_3 is the label from q_{rip} to q_j .

So, this will ensure that we get an equivalent GNFA. So, meaning G' will be equivalent to G . Any string that is accepted by G will continue to be accepted by G' and nothing more. So, basically, we have reduced one state, we have removed q_{rip} . Now we can do the same thing again. So, now we check. Basically, what we do is we make a recursive call of convert G' . Meaning for G' we do the same thing.

So, we check whether it is two states, if so, return R , otherwise we do the whole thing again. You identify a new state you remove and, so at some point you run out of states and we are, we end up with a 2-state GNFA in which case it is clear what is the regular expression. So, hopefully the outline is clear. We move from a DFA to a GNFA with two more states. And then we successfully reduce the number of states one by one, till you get a 2-state GNFA. And in a 2-state GNFA there is only one transition arrow. And that transition arrow contains some regular expression. And that regular expression is the equivalent regular expression for the current GNFA and for the previous GNFA and finally to the DFA that we started with.

(Refer Slide Time: 32:10)





Claim 1.65: For any GNFA G , convert (G) is equivalent to G .

Proof: If $k=2$, this is true by the definition of GNFA. If $k \geq 2$, we will show that G' is equivalent to G .

Suppose G accepts w . Let G go through the states $q_{start} q_1 q_2 \dots q_{accept}$.

→ If q_{rip} is not in the above sequence, then G' accepts w because all the transitions remain in G' .



Proof: If $k=2$, this is true by the definition of GNFA. If $k \geq 2$, we will show that G' is equivalent to G .



Suppose G accepts w . Let G go through the states $q_{start}, q_1, q_2, \dots, q_{accept}$.

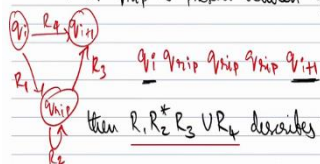
→ If q_{rip} is not in the above sequence, then G' accepts w because all the transitions remain in G' .

→ If q_{rip} is present between q_i and q_{i+1} ,

$q_i \xrightarrow{R_1} q_{rip} \xrightarrow{R_2} q_{rip} \xrightarrow{R_3} q_{i+1}$

in G .

→ If q_{rip} is present between q_i and q_{i+1} ,



This shows that $G' = \text{closure}(G)$ is equivalent to G . By using induction, we can reason that G' is equivalent to the final 2-state GNFA.

This shows that G is also equivalent to the 2-state GNFA and hence to R as well.

$R_1 R_2^* R_3$ then $R_1 R_2^* R_3$ describes this as well.

This shows that $G' = \text{closure}(G)$ is equivalent to G . By using induction, we can reason that G' is equivalent to the final 2-state GNFA.

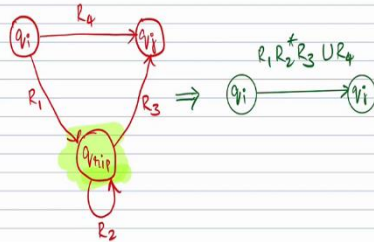
This shows that G is also equivalent to the 2-state GNFA and hence to R as well.

This completes the proof that regular expressions are another way to characterize regular languages.



Main ideas of the proof:

The key idea of the proof is in reducing the number of states of the GNFA by one.

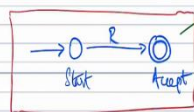


G is the GNFA



Suppose A has a 6-state DFA. This can be converted to an 8-state GNFA. Then we connect this into a 7-state GNFA, 6-state GNFA, and so on till we reach a 2-state GNFA.

2-state GNFA



R is the eqvt reg. exp.

Then we get $L(R) = A$.



Equivalence of reg. expressions and regular languages

Theorem 1.54: A language is regular iff some regular expression describes it. if and only if

This is yet another characteristic for regular languages. We prove it in two parts:

(1) IF (\Leftarrow) and (2) ONLY IF (\Rightarrow)

Lemma 1.55: If a regular expression describes a language, then it is regular.



So, now, one very small proof since we did not do much in detail. This is a claim in the textbook, it is presented as the claim as part of the, one of the proofs. So, what it is claiming is that $\text{convert}(G)$ is equivalent to G , meaning $\text{convert}(G)$ is the result of this process. So, it is a regular expression given a k -state GNFA.

It claims that $\text{convert}(G)$ gives you a regular expression which is equivalent to the GNFA G , meaning whatever string is accepted by the GNFA G is described by the regular expression and vice versa. So, the proof is fairly just an inductive argument.

If G has just two states, this is true by the definition of the GNFA, because there is only one arrow. If G has more than two states, so if k is greater than 2, then what we have to show is that, we will just show that G will be transformed to G' by reducing 1 state. We will just show that G' is equivalent to G , and then inductively, it follows G' is equivalent to the next one G'' or something and so on so on so on till we get a state of a GNFA of two states, at which case it is true by the base case of two states.

So, for now, we will just focus on this claim that G' is equivalent to G . This is what we want to show, meaning the exercise of removing one state preserves the language recognized by G . So, we will do something very simple. Suppose G accepts some string w , we will show the G' also accepts the same string. So, suppose G accepts w . Now let us say the in the process of accepting the string, it goes from the start state to q_1 to q_2 and so on till q_{accept} . And what we want to say is that G' also accepts the same string.

So, suppose q_{rip} is not in the above sequence $q_{\text{start}}, q_1, q_2$, this does not contain q_{rip} , in which case it is fine because for any pair, let us say q_i, q_j , there was some transition. The new q_i, q_j also contains the existing transitions because this is the transformation. q_i, q_j remains q_i, q_j but the label R_4 was replaced by $R_1R_2^*R_3UR_4$. So, any string that could take you from q_i to q_j earlier can still take you from q_i to q_j because any string that could take you from q_i to q_j earlier was described by R_4 .

Still R_4 is part of this regular expression. So, all the strings that are part of R_4 can still take you from q_i to q_j . So, if q_{rip} is not in the above sequence, all the transitions remain in G and are valid transitions. Hence G' also accepts w . Suppose this sequence contains q_{rip} somewhere. Just for sake of simplicity, let me say that q_{rip} appears between some two states in the sequence. Let us say the two states were q_i and q_{i+1} , these two states.

And let us say, q_{rip} appears more than once. Let us say q_i then q_{rip} , then q_{rip} again then q_{rip} once again before q_{i+1} . So, now if you notice the arrow from q_i to q_{i+1} would have been replaced by $R_1R_2^*R_3UR_4$. So, where what is, maybe I will just use a different color. Let us say this is q_i , this is q_{i+1} , this is q_{rip} . So, this is R_4 , this is R_1 , this is R_3 and this is R_2 .

Suppose this was the earlier sequence of states. Now, we used R_1 to go from q_i to q_{rip} , we used R_2 to remain in q_{rip} and R_2 again to remain in q_{rip} and then R_3 to move to q_{i+1} . So, $R_1R_2R_2R_3$ was used to move from q_i to q_{i+1} . Then, now in the present GNFA, the q_i to q_{i+1} label is replaced by $R_1R_2^*R_3$.

So, $R_1R_2R_2R_3$ is still part of this regular expression. Hence the same string can be accepted even though q_{rip} is not there, no longer there in the GNFA, even then this string is accepted as part of the transition from q_i to q_{i+1} . So, this shows that whatever be the situation, whether q_{rip} features in the sequence or it does not, the converted GNFA G' will still accept the string w .

So, what we showed is that if G accepts the string w , G' also accepts. And the same reasoning can be used in the opposite way. If G' accepts then it has to be either using $R_1R_2^*R_3$ or through R_4 . In either case there is an equivalent transition in the G as well. So, G' and G are both recognizing the same set of strings.

So, this we showed that, this shows that G' and G are equivalent and you can use the same argument again and again and use mathematical induction. So, G' is equivalent to the let us say G'' where one more state is reduced and so on and so on till we get to the final 2-state GNFA. So, the final 2-state GNFA, let us call it G_0 or G_2 or something. So, G_2 has these two states, and then it is immediately clear what the equivalent regular expression is.

So, at that stage, it is very clear what the regular expression is and that will allow us to tell what the equivalent regular expression R is. So, that is how we move from a DFA to a GNFA and then successively reducing the number of states while preserving the language recognized. And finally, we reach a 2-state GNFA and then the two-stage GNFA will tell us the regular expression that is equivalent.

So, I will just quickly summarize. The goal of this lecture was to complete the proof of this theorem which was that a language is regular if and only if some regular expression describes it. In the previous lecture, we already saw that if we have a regular expression describing some

language it is regular. So, now our goal is if a language is regular we should have a regular expression describing it.

So, suppose the language is regular which means there is a DFA that recognizes that language, we add two states, a new start state and a new accept state and we add empty set transitions to all the other places to get a new GNFA which is clearly equivalent. So, this means we are not adding anything.

Basically, if you had 6-state DFA, we will get an 8-state GNFA. And from that GNFA we successively reduce the number of states. So, by doing this operation again and again. We pick a state which is not a start or accept state, we remove it by relabeling the other arrows. So, for all the pairs q_i, q_j we relabel it by seeing what is q_i to q_{rip} , q_{rip} to itself and q_{rip} to q_j .

This needs to be done for all the pairs. Even the pairs q_i, q_j we have got in both order. And also, the pair q_i, q_i , and this will give us an equivalent GNFA but with 1-state less. And we keep doing that till we get a 2-state GNFA. So, a 2-state GNFA will be of this form. It has just a start state and accept state. And just one transition labeled by a regular expression.

So, for this 2-state GNFA, the equivalent regular expression is trivial. It is just the label of that one transition. And that completes the procedure of how we can get the regular expression for any regular language. So, from any regular language, you start with the DFA then move to the GNFA and then successively reduce the number of states till we get 2-state GNFA, and that tells us how to get the regular expression.

And with this, we have completed the proof that, so this completes the proof that regular expressions are another way to characterize regular languages. So, now the proof is complete. This shows that given any regular language, there is a regular expression. So, to show that the language is regular, we have a third way now. So, the earlier two ways were that you provide DFA or you provide NFA.

Now, one way to show, a new way to show a language is regular is to provide a regular expression for it. So, and, so this can be quite useful. So, if you are asked to show that a certain language is regular, instead of building a DFA or an NFA, we could just build a, or we could just construct a regular expression that characterizes that language.

So, that, with that, we complete Week 2's lectures. So, just to quickly summarize. What all did we see? We saw Non-deterministic Finite Automaton, we saw that, we could prove these are

equivalent Deterministic Finite Automaton. We saw that we could use the NFAs to prove the closure properties, that is regular languages are closed under union, concatenation, star. And then we explained regular expressions. And we showed that the regular expressions are also yet another equivalent representation for regular languages.

So, that completes Week 2. And, so we have regular languages that we have already seen and three different characterizations for them. Now we will move on to languages that are not regular. So, one may wonder, is everything regular? Can we represent all the languages using a DFA or an NFA or a regular expression? The answer is no and you will have to wait till next week's lectures to see some examples for that. Thank you.