


Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 38
Decidable Problems Concerning Regular Languages

(Refer Slide Time: 00:16)



Decidable Problems from Regular Languages

Theorem 4.1: $L_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \}$
A DFA is decidable.

Proof: The decider is the following TM.

M: On input $\langle B, w \rangle$, B is a DFA and w is a string.

1. Simulate B on w
2. If simulation ends in an accept state, accept. Else reject.

Hello and welcome to lecture number 33 of the course theory of computation, this is the beginning of week seven and also we are going to start chapter four of the Sipser book. So, right now, we are at the beginning of week seven, which is like exactly the halfway point of this course. So, in the previous lectures, so far, in week six, we saw Turing machines, we saw different types of Turing machines.

And then we saw search Turing thesis and then we said that a decider is intuitively an algorithm. So, we are going to try to understand what is computable and what is not computable? Or in other words, our goal is to try to understand what are the things? What are the computational questions for which we have an algorithmic answer or for which we can build a decider. And what are the questions for which we cannot build a decider.

So, towards that end, in this lecture, we will start with some languages and these languages or these problems as you can visualise them as problems; these problems will be based on other computational models that we have seen so far. So, first we will see problems based on DFAs and then problems based on NFAs regular expressions, context free grammars and so on.

And then slowly we will move into problems based on Turing machines, and then the undecidable problem. The first undecidable problem that we will see will be actually based on the Turing machine. So, first let us start seeing some examples of what I mean by problems based on regular languages or problems based on DFAs. So, the first problem that we will see in this lecture is A_{DFA} .

So, A_{DFA} means acceptance problems of a DFA. So, this consists of all the pairs, so it is a language where you are given a pair of $\langle B, W \rangle$ where B is a DFA and W is a string and you only accept those pairs that are where the DFA accepts the string. So, it consists of so A_{DFA} the language consists of all the pairs $\langle B, W \rangle$, where B is a description of a DFA and W is a string where the DFA accepts W .

And if you recall in the previous week, in the last lecture, we said that, given a DFA or an NFA or a Turing machine we could encode the Turing machine and I could give you a description of the Turing machine. So, here is the problem, so, when you see a decision question it is like this. So, I am giving you a description of a DFA and I am giving you a string and asking you, does this DFA accept this string? And that is the A_{DFA} question or in other words is the acceptance question, so, the A here stands for acceptance so it is an acceptance question of a DFA.

So, this is the first theorem that A_{DFA} is decidable. So, given a DFA and given a string there is an algorithm by which you can tell if this DFA accepts the string or not. We want everything to be like we are now going to see more and more from the lens of decidability. So, the algorithms have to be deciders meaning it has to hold regardless of whether the given string is in the language or not. So, if it is in the language it has to accept, if it is not in the language, it has to reject.

(Refer Slide Time: 04:08)

Decidable Problems from Regular Languages

Theorem 4.1: $A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \}$

A_{DFA} is decidable.

Proof: The decider is the following TM.

M : On input $\langle B, w \rangle$, B is a DFA and w is a string.

1. Simulate B on w
2. If simulation ends in an accept state, accept. Else reject.

Theorem 4.2: A_{NFA} is decidable.

Diagram: A DFA with states q_0, q_1, q_2, q_3 . q_0 is the start state. Transitions: $q_0 \xrightarrow{0} q_3$, $q_3 \xrightarrow{1} q_1$, $q_1 \xrightarrow{0} q_3$, $q_1 \xrightarrow{1} q_2$, $q_2 \xrightarrow{0} q_0$. The string 0110 is written above the transitions.

So, the decider is simple. So given a DFA and given a string, the decider the Turing machine what it does is? It knows it is given the DFA, so it knows the rules of the DFA and it just starts running the DFA on the input string. So suppose it knows that the string that is given is let us say 0110 something, so, now it knows the starting state, let us say q_{start} .

Now it sees that if you see if you read a zero, then you go to q_3 . And if you see a 1 from there, perhaps it is a loop. So you are q_3 again, and again the next is a one which is also so you take the same loop and come back to q_3 . And maybe the zero now takes you to q_2 , and then you check so you have read the entire string, you made the moves of the DFA. So I am just explaining how the DFA accepts, and then check whether this q_2 , whether it is an accepting state or not, this is how you check whether a string gets accepted or not. And the same thing can be executed by a Turing machine or an algorithm.

So if you tell the rules of the DFA and the string, the Turing machine can execute the DFA and check whether it ends in an accepting state or not. And if it ends in an accepting state you accept, if it does not end in an accepting state you reject, so that is it.

So you simulate the b the DFA on the string, and check where the simulation ends. So if it ends in an accepting state you accept otherwise you reject if it ends in a non accepting state you reject. So it is pretty straightforward. You just simulate the DFA and then see what the DFA decides whether it accepts or not, and then you reply. So there is no question of looping in the DFA because you are going to execute steps and at the end you are going to reach someplace. So there is no question of looping so it is a decider.

(Refer Slide Time: 06:15)

M: On input $\langle B, w \rangle$, B is a DFA and w is a string.

1. Simulate B on w
2. If simulation ends in an accept state, accept. Else reject.

Theorem 4.2: A_{NFA} is decidable.

$$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts } w \}$$

Proof idea: First convert NFA to DFA using epsilon-closure process explained in Chapter 1. After this the resulting DFA can be checked if it accepts w using Theorem 4.1.

M: On input $\langle B, w \rangle$, B is a DFA and w is a string.

1. Simulate B on w
2. If simulation ends in an accept state, accept. Else reject.

Theorem 4.2: A_{NFA} is decidable.

$$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts } w \}$$

Proof idea: First convert NFA to DFA using ^{my C}epsilon-closure process explained in Chapter 1. After this the resulting DFA can be checked if it accepts w using Theorem 4.1.

Theorem 4.3: A_{acc} is decidable.

So the next language is the acceptance problem of the NFA, so it is slightly more involved because a DFA is very clear, so you have a certain starting state and are given a starting state and a symbol. Let us say given a starting state, let us say q_s and there is 0. Now earlier, you knew that you are going to q_3 , now perhaps you can. there may be multiple outgoing arrows labelled 0, let us say that is a q_4 , there is a q_7 , all of them are labelled 0.

So, now there are multiple paths to track. How do you check whether a given NFA is the acceptance problem of an NFA A_{NFA} ? So, you are given an NFA and a string w and you have to decide whether this NFA accepts a string.

So, but then, so instead of trying to keep track of all the possible movements of the NFA, we can do something slightly more. Of course, this can be done also. But we will do something slightly different, but I think it is a bit more clever. So, if you recall, when we learned NFA the first time in chapter one, we also learned a theorem that says that NFAs are equal in power to DFAs or in other words, we saw that given an NFA you can construct an equivalent DFA.

So, if the NFA had let us say five states, the constructed DFA has two power five states. So basically, for every set subset of states of the NFA you had a state in the DFA, and then you made transitions accordingly. So again, I do not want to revisit the entire conversion process, but we saw such a result in the first chapter.

So, we can appeal to that result now, we can take the help of that result. So what I am saying is that first, so you are given an NFA B , we can convert B to a DFA. So, you can convert B to some DFA let us say C using this conversion process, the conversion process was a very clearly explained process.

So, we follow that process and convert the NFA to a DFA. And once we have done that, once we have converted it is an equivalent DFA. So, whatever strings are accepted by B will be accepted by this DFA and whatever strings are rejected by B will be rejected by this DFA.

So now, let us say this DFA, the converted DFA is let us say C is the converted DFA. Now, all we are asking is whether C is equivalent to B . So, now we are asking whether the DFA C does it accepts W ? So, meaning we are trying to, we have converted the problem of an NFA acceptance to a DFA acceptance. So, we have started with an NFA B . Now we are asking the question about whether B accepts w if and only if C accepts W . So, now, we have converted this problem to the first problem that is, we are asking we have converted to an A_{DFA} question.

So now, using that process that we just explained, we can simulate C on W and then decide whether C accepts W or not? So this is also decidable. So, given an NFA B and a string W , you first convert the NFA to a DFA C , and then use the process that we already explained to decide whether C accepts W or not and B accepts W if and only if C accepts W .

So, we check whether C accepts w and then say yes if it accepts w and say no if it does not accept. So, what did we do? We converted A_{NFA} to A_{DFA} so this is something that I am; this is a minor foreshadowing that I would like to do, because we will later formalise this notion of

converting one problem into another. So, what we did is? We took A_{NFA} and the A_{NFA} instance, so we were given an NFA and a string and we were asked whether this NFA accepts this string.

We converted to another problem where we have a DFA and a string and the answer to this original question was yes, if and only if the answer to this question was yes. So now we can as well answer the second question, but then we already know how to answer this question or the second type where we have a DFA (10:54). So, this is this process of converting one problem to a different problem this is called reduction and we will formally see this later in the fifth chapter, but I just want to, for those of you who are curious, you may just make a note of mental note of this and when you when we this reach chapter five, you will be able to see it more formally.

So, we converted A_{NFA} to A_{DFA} and now we know how to do A_{DFA} and hence A_{NFA} is decidable as well. The third thing is regular expressions, so the acceptance problem for a regular expression A_{REX} , so, same thing I am not going to elaborate much.

So, you are given a regular expression R and a string W , the question is does this regular expression generate the string W ? Why? Can we generate the string w from the regular expression? It is the same thing that works, we also saw equivalence of regular expression to NFA or regular expression to DFA. We can convert regular expressions to NFA and then use the decider for NFA. So, now, we have seen that A_{NFA} is decidable. So, by this decider for A_{NFA} we can decide a regular expression as well, so, a regular expression is also decided.



(Refer Slide Time: 12:16)

How can we check each and every string? ^{Not clear.}
Instead, we view the DFA as a graph. Is there a path from the starting state to an accepting state?

If there is a path, language is nonempty.

We can use BFS/DFS to check if an accepting state is reachable from the start state. If it is reachable, reject. If no accept state is reachable, then accept.

Exercise: Read theorems 4.1, 4.2, 4.3 & 4.4.





We can use BFS/DFS to check if an accepting state is reachable from the start state. If it is reachable, reject. If no accept state is reachable, then accept.

Exercise: Read theorems 4.1, 4.2, 4.3 & 4.4.

Theorem 4.5: EQ_{DFA} is decidable.

$$EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFA's, } L(A) = L(B) \}$$

Proof: Given A, B , we can use closure properties to construct a DFA C such that

$$L(C) = \underbrace{[\overline{L(A)} \cap L(B)] \cup [L(A) \cap \overline{L(B)}]}$$


Now, after three acceptance problems, we will see a slightly different problem called E_{DFA} . So what is E_{DFA} ? Given in description of a DFA, we are asked whether this DFA accepts any string or in other words we are asking, is the language accepted by this DFA the empty language, so, if you want to check whether there is even one string that it accepts okay. So, you may try to note that this is not of the same type.

So, earlier we got a DFA and a string and we had to just check whether this DFA accepts the string, but here we are asking does this DFA is the language accepted by this DFA the empty language? Or in other words is it any string that this DFA accepts?

Even if there is one string that DFA accepts, it is not the empty language because empty means it accepts no string. If it accepts no string then it is the empty language, so that is what we have to decide. So, maybe the first thought that comes to our mind at this stage is let us do this, let us start, let us try to try with all possible strings. So, we first try, let us say small strings of length one then length two then length three and so on we try each and every string. If any string is accepted, we say no, this is accepting some string. So, the language is not an empty language.

But then if it does not accept any string, then you will, there are infinitely many strings, so, you can have strings of length one two three four, but then there are strings of all possible lengths. So, at what point do you know that you are done? So, you may have to try out all possible strings and there are infinitely many strings, so, you will never be done.

So, if it is if it accepts, or if it recognizes a language that is not empty, at some stage, you see a string that is accepted and then you can say yes this does not accept the empty language, or you can reject the you can reject the DFA because it does not accept the empty language. But if it does indeed accept the empty language and recognize the empty language, then you are never going to get a formal confirmation in this process because whatever you try will not get accepted. So, when do you stop? That is the question. When do you stop? So that way, it is not very clear that unfortunately, it is not clear.

So we try a different approach. So what do we try is? See you have a starting state in the DFA and maybe there are some transitions which is 0 maybe there is 1 takes you somewhere maybe again 0 maybe self loop 1 maybe taking you somewhere else and so on. So, now, again 0 may take you here 1 may take you to some other state and so the point that I want to make is suppose there is an accepting state here and suppose there is a path to that accepting state, suppose this is labelled 0.

So, now, we know that any string the string 010 goes to this accepting state because there is a path here, suppose and we also know that this string 1010 goes to this accepting state, so 1010 takes this path. So, if there is a path from the start state to some accepting state you know it is the language accepted is not empty, for the language accepted by the DFA to be empty there should not exist any such parts.

So, maybe I will just use another colour for all the accepting states. There should not be any path that kind of crosses this barrier. All the accepting states should be on one side and there

should not be any way to reach the accepting states. So, basically we are just asking from the start state, is there a way to get to the accepting state, is there some path, some directed path that goes to the accepting state.

So, if there is a path, language is not empty. If there is no path the language is empty. So, we just want to say now, that the way is now, do not let us not look at it as a language or DFA or anything, we can just see this as a graph. So, the key idea is we just view this as a graph. So, this is the key idea, we just view the DFA as a graph and we are just checking whether there is a path from the first vertex that is the starting state to one of these designated vertices.

So, this can be done there are many algorithms, I do not want to get into each algorithm, but then if you go to an algorithms course, there are many algorithms you could do (17:46) search from starting from the start state and see whether you reach the accepting you reach one accepting state, you could do depth first search, because it is a finite state machine, DFA is a deterministic finite automata it has a finite number of states. So, you just run for some time and then you see which are the states that you can reach, and if it is reachable if an accepting state is reachable then the language is not empty.

So, you reject if no accepts it is reachable then the language is indeed empty and then which means you can accept. So, we have seen four languages all of which are decidable, A_{DFA} , A_{NFA} , A Regular Expression and EDFA and in the textbook you can check these are written as theorems 4.1 4.2 4.3 and 4.4, so, please have a look at these theorems go to this theorem perhaps some of them have gone through a bit faster maybe reading the theorems in from the textbook may help your understanding better, okay.

So, the next is EQ_{DFA} . So, here we are given two DFAs A and B and we are asked whether these two DFAs are equivalent. In other words, we are asked whether the language accepted by these two DFAs are the same.

So, in other words, so, if you think about it in the naive manner, you may want to you may be tempted to do this, try out first string the first small string and you are checking whether is it accepted by A suppose the answer is yes, is it accepted by B, now, if it is no then you can say the languages are not same because there is one string that is accepted by A but not by B.

But if it is both accepted, accepted with both A and B then you have to continue to the next string and then do the same. Suppose the next string is rejected by both again you have to go to the string after that and the string after that and so on. Till you find a string that is accepted

by one and rejected by the other, but then there is also not a never ending process like if the languages are indeed the same, you will never get a string that is accepted by one and rejected by the other.

So, that way we never are going to be able to reach a conclusion if the languages or if it is a yes instance meaning if $L(A) = L(B)$ you will just keep doing this but never end. So, instead we use another way. So, these are also again appealing to what we have learned in chapter one.

(Refer Slide Time: 20:40)



$EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFA's, } L(A) = L(B) \}$

Proof: Given A, B , we can use closure properties to construct a DFA C such that

$$L(C) = \overline{L(A) \cap L(B)} \cup [L(A) \cap \overline{L(B)}]$$

Symmetric difference of $L(A)$ & $L(B)$

$L(A) = L(B) \iff L(C) = \emptyset$

Proof: Given A, B , we can use closure properties to construct a DFA C such that



$$L(C) = \overline{L(A) \cap L(B)} \cup [L(A) \cap \overline{L(B)}]$$

Symmetric difference of $L(A)$ & $L(B)$

$L(A) = L(B) \iff L(C) = \emptyset$

On input A, B , which are DFA's.

1. Construct DFA C using closure properties.
2. Run the decider for EQ_{DFA} on C .

$L(A) = L(B) \Leftrightarrow L(C) = \emptyset$

On input A, B , which are DFA's.

1. Construct DFA C using closure properties.
2. Run the decider for Edfa on C
3. If the Edfa decider accepts, accept. Else reject.

Decidable Problems using CFL's

A dec = $\{ \langle A, w \rangle \mid A \text{ is a DFA that accepts } w? \}$

So, we know that regular languages are closed under union, intersection, complement, so, these closure properties are going to be very useful. This means that if there is a DFA let us say A and it recognizes his language $L(A)$, there is another DFA that recognizes the complement language that is $L(A)$ complement.

And if there are two DFAs A and B , then there is a DFA that recognizes the union of the two languages, there is a DFA that recognizes the intersection of the two languages. So that is what closure means. So, given A and B , I can make a DFA that accepts a union. I can make a DFA that accepts a complement everything and the intersection.

So, using these closure properties I can construct a DFA C that accepts the symmetric difference of $L(A)$ and $L(B)$. So, what do we mean by symmetric difference? That I have represented here pictorially, so, suppose $L(A)$ is a circle on the left, so, suppose this $L(A)$ and suppose $L(B)$ is a circle on the right, this is a Venn diagram, symmetric difference means the set of strings that are in $L(A)$ but not in $L(B)$, so, the intersection is removed plus the set of strings that are in $L(B)$ but not in $L(A)$. So, we are asked, we want the strings that are only in one language, but not in the other, so, in $L(A)$ but not in $L(B)$ and $L(B)$ but not in $L(A)$.

So, we are this will not include the strings that are in neither a $L(A)$ or $L(B)$ and this will not include the strings that are in both $L(A)$ and $L(B)$, so, this is the symmetric difference between this region and this region alone so the mutual intersection is skipped. And so, the symmetric difference is represented like this using union intersection and complement it is $\overline{L(A) \cap L(B)}$ so which is this part union $L(A) \cap \overline{L(B)}$ complement which is this part.

So, $L(A) \cap \overline{L(B)}$ is this, $\overline{L(A)} \cap L(B)$ is this and is the union of both and since DFAs are closed under regular languages are closed under complement intersection and union, we can construct such a DFA by all the closure properties. So, if you remember all the closure properties, we proved by constructively, so, to show that regular languages are closed under complement we explain how to construct a DFA that recognizes the complement language.

So, we can make such a DFA C that accepts the symmetric difference of $L(A)$ and $L(B)$. Now, when $L(A)$ is equal to $L(B)$, what is the symmetric difference? So if $L(A)$ and $L(B)$ are the same, there will be nothing that is in $L(A)$, which is not in $L(B)$ and vice versa. So, which means that if $L(A)$ is equal to $L(B)$ then the symmetric difference will be empty and if symmetric difference is empty, then $L(A)$ is equal to $L(B)$.

$$L(C) = [\overline{L(A)} \cap L(B)] \cup [L(A) \cap \overline{L(B)}].$$

So, all that we are asking is, so we know how to construct a DFA which recognizes a symmetric difference. Now, all that we need to know is whether the equality or the equivalence of A and B corresponds to the case when C recognizes the empty language. So this is yet another case where we are just referring to something that we have already done. We know how to decide whether a given DFA accepts the empty language. So now we constructed this DFA C sorry! C which recognizes the symmetric difference and all that we are interested in now knowing is if the symmetric differences, empty or not.

$$[L(A) = L(B)] \Leftrightarrow [L(C) = \emptyset]$$

So we check, we use E_{DFA} to decide. So we use the E_{DFA} decider to decide if $L(C)$ is empty. If $L(C)$ is empty we accept because if $L(C)$ is empty, we know that L is equal to $L(B)$ which means it is a yes instance. The question was whether the two DFAs recognize the same language? If $L(C)$ is not empty, then you reject it because then that means that $L(A)$ is not equal to $L(B)$ So, that is it. Even though I explained a lot, the procedure is very simple you construct a DFA C using the closure properties, run the decider for E_{DFA} on C , if E_{DFA} accepts we accept and if E_{DFA} rejects we reject.

And E_{DFA} the decider will certainly accept or reject because it is a decider it cannot loop and that completes the three languages or the (four) sorry! Five languages that I want to talk about which are decidable languages based upon regular languages. So, acceptance problem of a DFA NFA regular expression, the E_{DFA} asking whether a given DFA is empty, recognize the

empty language and the last one is equal is equivalence DFA asking whether the given two DFAs are equivalent, whether they recognize the same language and all of these are decidable.

And that is all that I have for lecture number thirty-three. In the next lecture, we will see problems from context-free languages. This lecture was about problems from regular languages, next lecture we will see problems from context-free languages. Thank you.