[SQUEAKING] [RUSTLING] [CLICKING]

PROFESSOR: OK, why don't I get started? So OK, what have we been doing? So last time, we considered a bunch of procedures for testing properties of various automata and grammars, the acceptance problem for DFAs, for NFAs, the acceptance problem, which is really degeneration problem for context-free grammars, and emptiness problems for DFAs and context-free grammars. And also, we showed that A,TM is Turing-recognizable.

There's a question here in the chat already about that. Yes, we did show that. That was the universal Turing machine that we presented at the end. That's what shows that A,TM is Turing-recognizable. We mentioned that A,TM is not decidable, which we promised we will prove today. And we will do so. OK, so that is the plan-- proving A,TM is undecidable. And we'll introduce the method for doing that called the diagonalization method.

I will also show the complement of A,TM is Turing-unrecognizable. Even though A,TM itself is recognizable, the complement is not. And then we will introduce another method called the reducibility method for showing other problems are not decidable and give one example of another problem, which is not decidable, assuming we have time to get to it. If not, we will delay that part until next Tuesday's lecture.

All right, acceptance problem for Turing Machines. So just as I mentioned, we showed that A,TM, which is the language of Turing machines and inputs where the machine accepts the input. We showed that was recognizable. We claimed it was decidable. Today, we're going to prove it's not decidable.

All right, now the method that we're going to use, which is really the only method out there for proving a problem as undecidable is called the diagonalization method. I mean, ultimately, we're going to show the reducibility method as well. But it really depends on having already shown some other problem undecidable via the diagonalization method. So ultimately, everything hinges on the diagonalization method, which is really what we have.

And so to introduce the diagonalization method, I'm going to make a bit of a digression into a branch of mathematics called set theory or it's a part of mathematical logic, where the method of diagonalization was first conceived of back in the late 19th century by a mathematician called Georg Cantor. And Cantor was considering the problem of, how do you compare the relative sizes of infinite sets? For finite sets, the problem of comparing-- somebody said that Cantor went crazy. That is true. And maybe I don't know why he went crazy. But he did go-- he had some mental problems, unfortunately.

And so how do we compare the sizes of sets in general? If they're finite sets, we can just count them up. We can say, whoa, this set has 11 elements, and the other set has 10 elements. So the one with 11 is bigger. Or if they both have 11, they're the same size. Well, that's not going to work for infinite sets because you can't count them up.

And so he had-- Cantor had the following idea for comparing the sizes of infinite sets. And that was, basically, to see whether you can have a function that would map from one set to the other set with certain properties. And those properties are called, traditionally, well, I mean, in the past, have been called the one-to-one and onto properties for the function. I'll tell you what that means.

But the concept is very simple. So a one-to-one function is a function that's mapping from A to B. Those are the two sets whose sizes we're trying to compare. And the function being one-to-one just means that there are no collisions. If you have two different elements of A, they're never going to map onto the same element of B. So two different elements of A always map onto two different elements of B. So that's the one-to-one property. It's also called injective.

And the other property is called onto or surjective, which is that the range of f has to be all of B. So you're not allowed to miss any elements, but you have to hit everything. And when you have both of those properties, the function is called a one-to-one correspondence or a bijection also, OK?

Now another way of looking at it-- I don't want to make this more complicated than it needs to be. It just simply means that two sets are considered to be the same size if we can match up the elements with one set with elements of the other set. You just pair them up. For example, well, if you have finite sets, that idea, that informal idea just works exactly as you would expect.

For example, if we have two sets-- here's a set of puppies. Here's a set of kittens. Now we want to show that those two sets have the same size. We could count them up, as I mentioned, and see that there are six elements in both. But counting up does not work for infinite sets.

So we can just match up the elements of the puppies with the kittens, and then we know we have the same number of puppies as kittens. OK, now that has the advantage of it making sense when you have infinite sets. So we're just going to extend that idea and apply it to infinite sets too. And then we'll have a notion of what it means for two infinite sets to have the same size.

And you might wonder, what do you get? Are all infinite sets of the same size when you use this notion or not? What happens? Well, some strange things do happen. But there actually are quite some interesting structure there that emerges. So anyway, I don't want to rush on. Questions on any of this? If you want to-- quick question. This, hopefully, was not too hard, but I want to make sure everybody's together with me on it, so we can pop in-- I'll give a few seconds for a chat if you have any questions.

The range of the set is all of the elements that you hit as you look at the different possible elements of A. So all of the things that f hits, the standard notion of a range of a function. So the range of f has to be equal to B. You have to hit everything.

All right, can you think of a one-to-one correspondence as a relabeling? Yeah, I'm not sure that's helpful. But, yes, you can think of as a relabeling of the elements in a sense, but yeah. I just think of it as a matching up.

All right, so let's move on. So coming out of this notion of sets being of the same size with this notion of countable sets, as we'll see in a minute, so let's do an example. Let's take the set of natural numbers-- 1, 2, 3, 4, dot, dot, dot, dot, and the set of integers, which includes the natural numbers but also the negative numbers and zero.

So the natural numbers are typically referred to as N. The integers are typically referred to as Z. And what do we think of the relative sizes of N and Z? Well, N is a subset of Z. It's a proper subset of Z. So you might, at first glance, think that Z is larger, and they're not really going to be of the same size.

But actually, it turns out that there's a very simple way-- the arithmetic and the properties of infinite sets can be a little bit surprising. And there is a way of matching up all of the elements of Z with their own elements of N. And so you can show that following that definition, that these two sets are, in fact, of the same size. So let's just quickly go make sure you see that.

So here is N. Here is Z I'm going to write down a table which shows how they match up. Here is the function f of n that I'm going to be describing. That's the one-to-one correspondence. And so here are the natural numbers-- dot, dot, dot, 1 through 7, dot, dot, dot. And here are the elements of Z that I'm going to be matching up. This is how the function is working.

So one, I'm going to-- f of 1 is going to be 0. F of 2 is going to be minus 1. F of 3 is going to be 1. And I'm just giving you a way to match up each of the natural numbers with integers so that every single integer has its own natural number and vice versa. So 4 goes to minus 2. 5 goes to 2. 6 goes to minus 3. 7 goes to 3, and then minus 4, and then 4, and minus 5, and then 5, and so on.

You're clearly going to cover all of the integers in this way. And each of the natural numbers is going to have its own integer. And there's never going to be any collisions. There's never going to be two natural numbers assigned to the same integer. So this meets the conditions of a one-to-one correspondence and shows that the natural numbers and the integers have the same size following this definition.

OK, let's do one that's slightly more complicated and, perhaps, slightly more surprising, which is that if you consider all of the rational numbers, and then they have the same-- that is a collection. Even though the rational numbers seem to be much richer and more numerous than the integers, from this perspective, they have the same size.

And for the simplicity of my presentation, I'm going to consider only the positive rational numbers, which I'm going to write as Q+. So those are all of the positive rational numbers that can be expressed as a ratio of two natural numbers. And I'm going to show that there is a one-to-one correspondence between these positive rational numbers and the natural numbers.

OK, so here, I'm going to, again, make a table, just as I did up above-- so comparing the natural numbers and the positive rational numbers. And to do that, I'm going to write down over here on the side, just to help you see how I'm getting this table, a separate table that has all of the positive rational numbers appearing in a nicely organized way.

Here are all of the rational numbers here that have 1 as a numerator, that have 2 as a numerator, and so on, and going through across the columns, the different denominators. So the numbers inside here represent all of the different rational numbers. And so whatever rational number you have, m over n, that's going to appear down the nth row in the nth column. That number is going to appear. So they're all going to show up.

And I'm going to use this table here to fill out this function. So here are all the natural numbers. And the way I'm going to assign the rational numbers to appear over here is I'm just going to work my way in from the corner. And I'm going to do that by doing layers. So first, I'll take the 1 in this number here in this corner. Then, I'll do these three that surrounded it, and then these guys that surround that, and these guys sort of in shells going around the previous ones that I've already covered. OK, so let me illustrate that.

So here's 1 over 1, my first rational number that I'm going to enter into the table, appearing right over there. So next, I'm going to go 2 over 1. That's going to appear in the table over there. Now, we have 2 over 2. Now that's actually a little bit problematic because 2 over 2 has already been done. And if we put that in-- because 2 over 2 equals 1 over 1. They're both the equal to the rational number 1.

So if we put 2 over 2 in this table over here, then we would no longer have the one-to-one property because both 1 and 3 would be mapping to the same point. So we're just going to simply skip over 2 over 2. I'll show that as kind of graying it out. So we're not going to add that one on to the table onto my function.

But so we'll skip over that. We'll go to 1 over 2, which has not been seen before. And then we're just going to continue doing the same thing, now going to this next shell out here. 3 over 1, 3 over 2, 3 over 3-- same thing, we're going to skip over that one-- 2 over 3, 1 over 3. I hope you're getting the idea. So I'm not going to fill this table. I ran out of room to fill out this table some more.

But just to look at how the process is going to continue here, we get 4 over 1. Now 4 over 2, again, is a number we previously seen because that's the rational number 2. We saw that up here when we had 2 over 1, so we're going to have to skip that one. 4 over 3, 4 over 4-- that was going to skip. 3 over 4, and so on. OK? So by following this procedure, I'm going to be able to define this function. Now this function is not particularly nice in terms of having an elegant, closed form.

But it is a totally legitimate function in the sense of being a mapping from natural numbers to the positive rational numbers. And it has the one-to-one correspondence property. So it pairs up each of the natural numbers with each of the positive rational numbers. They each get their own mate, in a sense. And so that shows that the rational numbers, despite the fact that they're dense, and they have all sorts of very more much bigger seeming, they really, from this perspective of the sizes of the sets, they have the same size as the natural numbers do.

And so with that, it suggests that the following definition that a set is countable if it has the same size as the natural numbers or it's finite. From this perspective, we're going to be focusing on infinite sets. But yeah, countable or countably infinite, sometimes people say, if it has the same size as the natural numbers. Or otherwise, you have to include the finite sets as well.

And countable means you can just go through the elements of the set as a list. So you can count them-- the first one, the second one, the third one, the fourth one, dot, dot, dot. And once you can do that, and that counting is going to hit everything, then you know can match them-- you can pair them up with the natural numbers. And so therefore, you have a countable set. OK.

So as we've shown, both Z, the integers, and the positive rational numbers are countable sets. OK? Now you might think that, well, since we're allowing ourselves to do something as arbitrary, in a way, as this kind of a function to match up the natural numbers with whatever set you're trying to measure, that every set is going to be countable, if you think about it that way because it seems like you're allowing too many possibilities.

And then all the infinite sets are going to end up being the same size as the natural numbers. Well, that is, in fact, is not true. And I don't know. Cantor, is the one who discovered that. I don't know if that was surprising or not. But it is interesting, I think, that there are different sizes of infinities.

And so we're going to now take a look and see that. But I want to, again, I want to stop and make sure we're all together. Can we always find a closed formula for f, somebody's asking me. I don't know. For this particular f, you probably could, but it would probably be very messy. But in general, that's not a requirement, having a closed form for the mapping function. Any function is allowed as long as a one-to-one correspondence.

Are we all together on this? Are we comfortable with the notion of some infinite sets having the same size as the natural numbers, and therefore, we're going to call them countable sets? And we're going to show some other sets are going to have more elements. They're going to be uncountable. They're going to be beyond-- strictly speaking, strictly larger sets in the sense that we won't be able to put them into a one-to-one correspondence with the integers. So is N the smallest infinity? Yes, N is the smallest one.

So any infinity is going to have-- you're going to always-- I don't think you even need it-- you need a special-- whenever you have an infinite collection, you can always find a subset which is going to be a countable subset and is going to be the smallest of the infinities, but there are bigger ones. All right, why don't we move on?

OK, so the example of a set that we will show is not countable-- an uncountable set, as we call it-- is a set of real numbers, which we all know what these are, I hope. These are all of the numbers that you can express by possibly infinite decimal representations like pi, or E squared of 2, or any of the other familiar ones. Rational numbers are also members of or also count as real numbers and integers too. And so but there are these additional numbers that you can get by the decimal expansions which may not be rational.

And that collection, even though, in some ways, it looks somewhat similar to the rational numbers, the real numbers-- I hope I said that right. The real numbers, which is the set I'm considering here, the ones with the infinite decimal expansions, they actually are much larger. So the theorem is-- and this was discovered by Cantor-- that R is an uncountable set. And the reason why I'm introducing this is, besides that I think it's interesting and has some relation to the kinds of things we're doing, but it's really for the purposes right now, is to introduce this method called diagonalization.

OK, that's what we're going to use later on again. But this is the first time it appeared. So we're going to use a proof by contradiction in order to show that R, the collection of real numbers, is uncountable. And we'll assume, for contradiction, that R countable. OK?

So if we assume that R is countable, it means we have, by definition, a one-to-one correspondence from the natural numbers to the real numbers. OK, so we can match up all of the natural numbers with the real numbers in a one-to-one and onto fashion. So we can pair up the natural numbers with the real numbers. And we will cover all of the real numbers by doing that. And we can make a table, just like I did before. Here it is.

And you can fill this out with all of the real numbers. And that was what the assumption means. And I'm going to show you that that's impossible. Something is going to go terribly wrong if you do that. Now you might disagree. You might be surprised. You might think, well, Professor Sipser is wrong, that I'm going to work on this overnight and forget the rest of my classes, and I'm going to come up with a list of all the real numbers. I'm going to fill them out here and show that that's impossible.

So let's say, hypothetically, just for my example's sake, here is the list that you came up with. So you have to match up something with the 1. Let's say, E. That was a special case, so you're going to stick that with 1. And then pi came out to be the number that you connected up with 2. You understand what I'm doing here? I'm making a list. I'm trying to make my correspondence-- my matching up between the natural numbers and my real numbers that I'm hypothesizing to exist for a contradiction.

So 3, I don't know, 3 is gets connected to 0. I'm making this up, obviously-- not on the spot. I wrote the slide, you know, yesterday. But here is the square root of 2 showing up for whatever reason. Here is 1/7. Here is some other number, which, I'll be interested if you recognize that one. That's a subtle one. This is 1.234-- yeah, some familiar looking sequence here. And whatever it is-- whatever it is, this is what you came up with.

You claim that this is going to work as-- very good. Somebody got it. It's i to the ith power. But let's not get hung up on that, please. I to the ith power, oddly enough, can be seen as a real number under-- if you define the imaginary exponentiation, which is weird.

But anyway, let's not get too distracted. So here's my list of numbers, my real numbers that I've matched up in my table here with the natural numbers. Now I claim something goes wrong. What goes wrong? I'm going to show you that you actually did not do as you claimed, that there's actually at least one number that's missing from this list. And I'll exhibit that number. I'm going to show you what that number is. I'm going to explicitly come up with a number and show you that there's that number it's missing from the list.

So here, it's going to be a number x. X is the one that's missing. So how am I going to get x? So x, well, I'm going to start it off with 0 point, and then I'm going to fill out the rest of the places. And how am I going to get those places? I'm going to look at the table here.

So I want to look at-- to get my first digit after the decimal point of x, I'm going to look at the first digit after the decimal point of the very first number. So I take the first number, look at the first digit after the decimal point, which happens to be a 7. And the number I'm going to put in for x It's not going to be 7. It's going to be anything except 7. Let say, 8.

For my next digit of x, I'm going to look at-- so my second digit of x, I'm going to-- everything's going to be up to the decimal point now, so I'm not going to keep saying that. I'm going to look at the second digit of the second number, which is a 4 after the decimal point. There it is. And for the second digit of x, I'm going to use something which is anything besides 4-- 5. I have some flexibility here.

So I could have used 6. I could have used 7. For those of you who have seen this before, who are going to nitpick on me, I'm going to stay away from 9 and 0 just because there is some little edge cases that arise there, which I don't want to get into because I don't think it's interesting for this argument. But since I have some flexibility, I'm going to avoid zeros and nines-- probably just nines is all I really need. And then the argument is just going to work fine, OK?

So the next digit is a 0. The third digit, or the third number is a 0. The third digit of x is going to be anything except for 0. It could be a 1. Then I have a 2 here. Anything except a 2, 6. Then a 5-- anything except a 5, a 1. A 9-- anything except a 9, an 8, and so on. There's an 8. There's a 2, and dot, dot, dot. I'm just going to keep doing that.

And I'm going to say you missed that number, whatever it is. But that number, that's a number. After I'm all done, it's going to be the decimal representation of something. And that number, I claim, is absent from the table. And you might say, well, it's really there. It's just on the 23rd row. You just didn't get to it and your slide. But it's there. Well, I'm going to say I know it's not in the 23rd row because it differs from the number in the 23rd row at the 23rd digit because I constructed it that way. This number is designed explicitly to be different from each of the numbers that's on the list.

So it can't be on the list because it's been constructed to be different from each of them in at least one place. So I think this is a beautiful argument, and it shows that no matter how you try to come up with a one-to-one correspondence, you're going to fail. You might say, oh, like, just one number? I did so much hard work. I'm missing just one number. Can't I get partial credit and put this one on the list now I fixed it? No.

Obviously, there are many, many numbers that are missing. If you put this one on the list, I was going to construct another number that was missing. So there's just not no possibility of fixing this. And in fact, the real numbers are uncountable-- cannot be matched up with the natural numbers. And there's nothing. It doesn't even come close.

So here, a summary of what I just said. F is not a one-to-one correspondence no matter what you try to do. You can't come up with a one-to-one correspondence. That's the contradiction. So that proves that R is uncountable. And that's why, by the way, we call this a diagonalization because we're going down this diagonal here that's where the term-- that's where the name is come from.

OK, so I'm happy to-- [CHUCKLES] OK, that's a-- somebody's asking me-- [CHUCKLES] I'm actually-- I have a-- is it here? I can't remember if it's here. I have a check-in coming about this. And somebody is anticipating that by asking the actually rather famous question about there being a possible infinity in between the natural numbers and the real numbers. We know the real numbers now are bigger than the natural numbers. But is there something that's an in between? I mean, this is a very, very famous problem, which I'll talk about when we get to our check-in, which is coming up.

Somebody is objecting just the way I've defined x using this procedure that really can't, in a sense, state, you know-- but x is a number. X is the result of what this procedure is. Following this procedure, we're converging on some particular value. And so that is a value. If you want, we can make a more precise way of determining. We don't have the flexibility of choosing the way I did in my example. We can make a precise procedure for coming up with these digits.

But I don't think there's anybody thinks there's anything that's-- there's any shortcoming in this argument in terms of the way we're defining x. I think it's worth understanding this because it's really going to set the groundwork for our proof that A,TM is undecidable, which is a little, I think, perhaps, slightly more abstract in a way in the way it sort of comes across. I'm going to try to connect the two. But I think it's helpful to understand at least this argument because this argument is diagonalization in its most raw form.

All right, I think we're good. So why don't we continue, then? So there are a number of corollaries that follow from the statement that the real numbers is uncountable. First of all, if you let script L be the collection of all languages, if you want to consider it over some particular alphabet, that's fine. But that's not going to be really important for this point that I'm going to make. So script L is the collection of all languages. So every subset of sigma star-- every subset of sigma star is a language. So look at all those possible subsets. So that includes 0 to the k, 1 to the k, plus every other language you can ever think of and more-- all possible languages.

So the collection of all languages is uncountable. There's uncountably many different languages out there. I don't want to belabor this point. You can just take this if you don't quite follow the quick argument I'm going to make here. But you can make a one-to-one correspondence from all languages to the reals so that each language gets its own real number. And the way I'm going to think about that-- let's put the real numbers into binary form.

And if you imagine here being sigma star, all of the possible strings of sigma star written out in their standard order. And now if you have a language here, A, it's just some arbitrary language. So that's going to have some of the strings of sigma star appearing. Like 0 is appearing, but 1 is not appearing. 00 is appearing and 01, but not these three. And I'm going to associate to A, my language, some real number in binary by putting in a 0 in the decimal representation-- well, the binary representation, I should say-- for that string if it's not there and a 1 if it is there.

And so a real number-- because there's infinitely many yes/no choices in the binary representation can represent a language because of each of the yes/no choices of a string being present or not. I'm going to put a 1 for when the string is present, a 0 when it's not present. So each language has its own real number, and each real number is going to be associated to its own language. Here, I'm restricting myself to real numbers between 0 and 1. That's not going to be an essential point. So let's not get hung up on that.

So the fact that the languages can be putting it into a one-to-one correspondence with the real numbers shows that the collection of all languages is also uncountable. Now another observation here-- another point worth noting is that if you look at sigma star itself, the strings, all possible strings, that's a countable set. The collection of all possible languages is uncountable. But the collection of all possible finite strings is countable because I can just list them. Here's my list of all possible strings, which you can put into a table if you like to think of it matching up with the natural numbers in that way.

Now I'm trying to make a point here, which is that if you take M, which is all possible Turing machines-- script M is all possible Turing machines-- the collection of all possible Turing machines is countable. There are only countably many different Turing machines. And you can argue that in all sorts of messy different ways. But I think the most simple way to see that is to think about each Turing machine as having a description, which is a string. So the collection of all descriptions of Turing machines is just a subset of sigma star, which we already know is countable. And the subset of any countable set is going to be countable.

So anyway, I think it's worth remembering that the collection of old Turing machines is countable. Whereas the collection of all languages is uncountable. And that tells you right there that some language is not decided because there are more languages than Turing machines. We've unaccountably many language, only countably many Turing machines, so that's fewer Turing machines than languages. There's no way to map all the languages onto Turing machines. So there's going to always be some languages that got unmapped. And so, in particular, there are going to be some languages which are undecidable. There are going to be some languages which are not Turing-recognizable. And anything based on some automata kind of a definition process is going to be some languages that they're not going to be defined.

OK, now what this corollary shows you that there is some language out there, which is not decidable. What we're going to show next is that there is a specific language-- A,TM-- which is not decidable. And but first, I think we have a check-in coming up. And let me give you a little bit of background here because this is relevant to this question that I got about intermediate size sets. So the question of whether there is a set of size between the natural numbers and the real numbers strictly in between-- so bigger than the natural numbers, not the same size as the natural numbers, but not the same size as the real numbers either, but in between in size. That was Hilbert's question number 1 out of his list of 23 that I talked about a few lectures back.

It's interesting that he put it as number 1 in that very privileged special place because I know Hilbert was very-- he felt that the understanding infinity was a really central issue in mathematics. And that if we can't answer a question like this, we don't really understand infinity. You want to understand what kind of sizes of infinities are there. We know there's the real number is bigger than the natural numbers. Is there something in between? So fundamental, really.

But it was shown during the course of the 20th century, really, in two separate steps-- one in the 1930s by Godel, one in the early 1970s by Cohen-- that we can't answer this question by using the standard axioms of mathematics. The answer can go either way. And both of them are consistent with the axes mathematics. So you're never going to be able to prove that there is a set whose size is in between or that there is no such set. It's just impossible to prove either way using the standard axiom of mathematics, which, actually, is kind of remarkable.

And so my question for you is-- and this is really a philosophical question, not one that is directly you need to know about material in the course. I think it's just a matter of your own interest. I hope you find it interesting. If you don't, you can just answer it randomly. But what's going on here that we can't answer that question about whether there is a set of intermediate size? Is it because our axioms for mathematics are inadequate?

Or maybe there is no such thing as a mathematical reality. You can talk about what's real here? What's the reality? Either there is a set in between or not. If you can imagine, all of these things have their own reality to them, well, then, there's going to be an answer. And then you would expect, well, maybe we can find better axioms, which will actually give us that answer. Or you can say, well, there is no reality.

And infinite sets are kind of human constructs anyway, so we can make them kind of play out any way we like. Mathematicians argue about that to this day. And it is, as I say, really, it's a philosophical question. But just out of curiosity, let's see how you guys end up deciding on that one.

So here is the poll. 5 seconds to go. Please vote. And we're going to end the polling, 1, 2, 3, now. All right, here we go. So there's no right answer. I think if most mathematicians were to, I think, the instinct of most logicians, especially, I'm not sure if general mathematicians really even care about this question, but logicians would probably have an instinct that, probably, there are sets in between. There's no reason that there shouldn't be. It seems kind of strange that there should be this sort of jump from the natural numbers to the real numbers and why nothing in between? But I don't think that question is really settled.

All right, let's continue on. I think we are-- OK, so we're going to-- our coffee break is coming, in case you're wondering. So this is my last slide before then. But this is an important slide, so please hold out. So here is our promised theorem of the day. I'm going to show that A,TM is not decidable, the acceptance problem for Turing machines.

And it's all going to be contained on this one slide. We're going to give a proof by contradiction using diagonalization. And we're going to assume some Turing machine, H, decides A,TM. And we're going to get a contradiction. So let's, first of all, make sure we understand what H is doing.

So H gets an input-- a Turing machine and an input. And H is going to be a decider, so it always halts with an accept or a reject. It's supposed to accept if M accepts w and reject if it doesn't. So in other words, it's going to reject if M rejects w either by halting or by looping. That's the job of H. And we're assuming we can do that.

But we will see a contradiction occur. So the way we're going to do that is really kind of just one step here in a way. And we're going to use H to construct another Turing machine D. H is going to be a subroutine to D. We've already seen us doing that in the past.

D is going to do something a little strange, just to warn you. D's input is just a Turing machine-- no w. And what D is going to do using its subroutine H is going to simulate H on input M, comma, the description of M. Now what is that?

Well, the description of M is just some string. So what H is trying-- what it's asking H to tell, to answer is, does M accept the string representing M's own description? It's as if we're feeding M into itself, which seems like a totally twisted thing to do, you might say. Why would you ever feed a program into itself? Somebody has written cannibalism here-- yeah, kind of. I'd say it's worse [LAUGHS] because it's not eating somebody else. It's eating yourself.

OK, but I claim that there are actual cases in practice where we do this. We feed programs into themselves. And the example that I know of where this is done is when you're making a compiler. You might want to make a compiler and then written in the same language that it's compiling. And then you feed the compiler into itself.

You may say, why even bother because it's already, obviously, if it, once it's running, you don't need to compile it again. But actually, an example where this was really used was when there was an optimizing compiler, I think, for C written on early Unix machines. And the optimizing compiler for C was written in C. So you would feed the optimizing compiler into the regular compiler, first of all. Now, you had the compiler running, but it was an optimized.

So but now that it's the optimizing compiler is running, you can feed the optimizing compiler into that, which is itself. Now you have an optimized optimizing compiler. So it really makes some-- there is at least one case where this has actually been done in practice. Not that we really care. This is a theory class-- but just for fun to observe that.

So here, H is trying to say, well, does in a M end up accepting when it's fed the description of itself? You know, at least, mathematically speaking, that's a reasonable thing to ask. And then what D is going to do when it gets the answer back from H-- H is a decider, don't forget-- is D's going to do the opposite of whatever H does. It's going to accept if H rejects and reject if H accepts.

So let's, in summary, let's pull this together, so it's easy to understand, in the end, what is D doing? D is going to accept. D is also going to be decider, by the way. So D is always going to either accept or reject-- Just. The opposite of what H tells it to do. So D is going to accept M exactly when M doesn't accept M because when M doesn't accept M, H is going to reject, and so then D is going to accept.

So D accepts M if and only if M doesn't accept M. That's exactly the condition in which D accepts M. I think it's important to just step back and make sure you understand this line because we have only one line left to get our contradiction. Right? Are we together? D accepts M if and only if M doesn't accept M. That's just by the way we've defined setup D.

Now what we're going to do is feed in, instead of M to D, and not some arbitrary feed, we're going to use feed in D into D. And that is going to be our contradiction because D is now going to accept D if and only if D doesn't accept D, and that's certainly impossible. That's our contradiction which proves that H cannot exist. And therefore, A,TM is undecidable.

OK, so we're done, except for the one point, which is that why is this a diagonalization? And I think you can get that from the following picture. If you imagine here writing down all possible times-- I'm going to make a table here. Here is the list of all Turing machines, including D, which is a machine which I built under the assumption that H exists. So D appears here somewhere. But here are all the other Turing machines. And here are all of these descriptions of the Turing machines along the labeling all of the columns.

OK, so these are the rows labeled. These are the column labels. And inside, I'm going to tell you the answer for whether a given machine accepts a given input. So for example, M1 accepts its own description but rejects the description of M2, but accepts the description of M3. I don't you know. I'm obviously I'm making this up. I don't know what M1 is. But just hypothetically, that's what the machine M1 does.

So I'm just filling out this table. H could get the answer to any of the elements in this table because it can test whether M4 accepts the description of M3. So H could fill out this table. So maybe M2 is a machine that always rejects. It's a very unfriendly, rejecting machine. M3 is a very friendly machine. It accepts all inputs. M4 rejects some and accepts others, dot, dot, dot.

Now, I want to look and see, what does D do? Now based on the information that I've already given you, we can understand what D does. So for example, what does D do when I feed it the description of M1? What does D do?

Well, we can look here. D accepts M if and only if M doesn't accept M. So D is going to accept M1 if and only if M1 does not accept M1. Well, M1 does accept M1. So D does the opposite. D rejects. So OK, I'm highlighting here. D rejects because D is going to do the opposite of what the machine does on its own it on input. So D on M2, you have to look what M2 does on M2. It rejects, so D does the opposite of that. It accepts.

And similarly, each one of these things-- D's answer is going to be the opposite of what the machine does on its own description, just by virtue of the definition of D. OK, and so on-- so far, so good. But the problem is, what happens when D is fed itself? Because, as you can see, we're heading for trouble because this is a diagonal down here. D is just one of the rows. That diagonal is going to intersect that row at this point. And D is defined to be going to be doing the opposite of what that element is, but it can't be the opposite of itself. And so that's the contradiction.

So I think we're-- I'm going to call us, give us a little break here. And then you can also text me in the meantime. I'll be happy to answer some questions during that. A little over 4 minutes to go-- so let's see. Let me see if I can answer your questions.

OK, what's so special about A,TM that enables us to do this? Why can't we do this on ADFA, for example? That's a good question. And the answer is that, in a sense, we can do this on DFA. I mean, I think this is, perhaps, a bit of a stretch. But DFAs could not answer ADFA. I mean, we could prove that in other ways as well by just-- we could use things like the pumping lemma, and it's not clear, even how you'd formulate ADFA.

But what's important here is that it's really the model talking about itself that really is where the problem comes up. So if you try to push this argument through to show that ADFA is not decidable by Turing machines, you're going to fail because we're starting off with a Turing machine. And I think I'm going to confuse myself if I try to just repeat it. But you won't get a contradiction because the Turing machine is not a finite automaton. OK.

Will this argument get into self-loops? I don't see why it would-- there is some self-reference, perhaps. We're going to talk about that a little later. So we're going to come back and revisit this argument in a week or so when we talk about the recursion theorem which talks about machines that can refer to themselves. But this is a little bit of a head-- getting ahead of ourselves.

So somebody's commenting on this reminding them of the barber paradox, if you remember that, which has some similarity. There is a town in which there was a barber which shaves every man who doesn't shave themselves. It seems he's a very good barber. So there are some people who shave themselves. And all the rest, the barber shaves. The question is, does the barber shave himself? Because he shaves only those men who don't shave themselves. So you've got a same kind of a contradiction. There is a relationship there.

So someone wants to know, where did we use the decidability? So we used the decidability to come up with H. Once we know that A,TM is decidable, then we have that H function, and then we can build D. So that's the chain of reason. So you assume A,TM is decidable. Then you have the decider called H, and then you can build D. Somebody wants to see the previous slide. What part of the slide do you want? So I'll leave that up there.

Why can we apply the proof that all Turing machines are accountable to all languages? Well, because Turing machines have descriptions. General languages don't have descriptions. And so that's why. OK, the candle has burnt to the bottom, and it's time to move on.

So now, let's look at the acceptance problem for queue automata. I'll give you a queue automaton on input, and I want to know, does it accept the input? Is that going to be decidable? And you have your choices. It's either yes, it is decidable because these are similar to pushdown automata and APDA is decidable, or no because yes contradicts results that we know at this point, or we don't have enough information to answer the question. OK, let's put that up. One second-- [LAUGHS] all right, that's it. Ready, going, gone.

So yes, the answer is, well, no. [LAUGHS] The answer is, indeed, the answer is B. True, that queue automata are similar to pushdown automata, but all these automata are similar to each other, and that's not going to be good enough.

What the homework has asked you to do is to show that you can simulate Turing machines with queue automata. So if you can answer the question about whether the queue automata will accept their input, that would allow you to be able to answer questions about whether Turing machines accept their input. And we just proved that's not possible. So it would be a contradiction if we could answer-- if we could decide A, queue, A.

Now, we have an example of an undecidable language. Let's look at an example of an unrecognizable language. Now A,TM is not going to serve that purpose because A,TM is Turing-recognizable, as we pointed out by the universal Turing machine. So A,TM is undecidable, however. How about an unrecognizable language?

For that, we will see that the complement of A,TM will serve. So the complement of A,TM is neither decidable nor even recognizable. Now it's not Turing-recognizable. And that's going to follow from a pretty basic theorem that connects recognizability and decidability that I've put up here on the screen, which is that if you have a language where it and its complement are both recognizable, then the language turnout turns out to be decidable. In fact, a language and its complement are decidable. But being decidable is closed under complement, so that's something you should be aware. But being-- OK. We'll get to that in a minute. But if-- so anyway, so if you have a language and it's complement both recognizable, how do we know the language is decidable?

So first of all, let's take the two Turing machines, M1 and M2 that recognizes A and A-complement. And we're going to put those together to get a decider for A. And that's going to work like this. It's going to be called T. So T says, on input w, what it's going to do, it's going to feed w into M1 and M2 both. A is the language, by the way, yes. A is-- when I say it's Turing-recognizable, you know, Turing-recognizable only applies to languages. So yes, A is often this symbol I'm going to use for languages, sometimes for an automaton. But A is typically going to be a language.

So now, I'm trying to make T be a decider for A from the recognizers for A and A-complement. So I'm going to take an input to T and feed it into both recognizers, M1 and M2. OK, I'm going to run them in parallel. What's nice is that because M2 recognizes the complement of what M1 recognizes, every string is going to be accepted either by M1 or by M2 because every string is either an A or an A-complement. So if I run M1 and M2 on w until one of the halts or one of them accepts, I know I'm not going to run forever because, eventually, one or the other one have to accept.

So and then I got my answer because if M1 accepts, then I know I'm in the language. But if M2 accepts, I know I'm in the complement of the language, so I'm out of the language. So if M1 accepts, then T should accept. But if M2 accepts, then T should reject. , So that proves that nice little theorem written at the top in blue. So I got my decider for A built out of the recognizers for A and A-complement.

Now immediately, it follows that the complement of A,TM is not Turing-recognizable because we know that A,TM itself is recognizable, but it's undecidable. If the complement was also recognizable, then A,TM would be decidable, but it isn't. So when something is decidable, either it or its complement have to be unrecognizable. And in the case for A,TM, it has to be the complement because we already showed that it is itself is recognizable. So that's the proof of that.

So here is a little picture of the way the world looks right now if you have here, in the middle are the decidable languages-- so these are all languages, this Venn diagram of languages. We showed earlier, the regular, the context-free, decidable, recognizable. Here, I've got the recognizable and what I'm calling the co-Turing recognizable. This is the collection of all complements of recognizable languages.

So A,TM bar, A,TM complement is the complement of a recognizable language. This region here are all the complements of the recognizable languages or the so-called co-Turing recognizable languages-- complement of. So A,TM is on this side. A,TM-complement is on that side. But if something's in both, by virtue of this theorem here, then it's decidable.

OK, last check-in for the day. From what we learned so far, which closure properties can we prove for the class of deterring recognizable languages? Choose all that apply. Well, as I say, you don't have to get it right. Let's not spend too much more time on this because we'll talk a little bit about it. Almost all-- its closed under almost all of them, but not all of them. Because-- are we done here? I think we're done-- 5 seconds. OK, here we go, ending polling.

I'm not sure what the meaning of [LAUGHS] deleting your answer is here. Everybody likes union, I guess. They're closed under all of these operations except complement. But we just proved it's not closed under complement, so I'm a little puzzled by why we have so many votes for closure under complement. We have here, A,TM is Turing-recognizable, but A,TM-complement is not Turing-recognizable. it's right here on the slide. I'm not trying to make you feel bad, but I'm trying to just point out that you think, please.

So now closure under union and intersection, I mean, you could kind of get those answers just by running things in parallel the way we did the proof here. You just run both machines. And if they both give-- I mean, it's a little tricky, I suppose. If either one of them accept, then you can accept. Or if they both accept, you just wait until they both have accepted. Otherwise, you just keep running. So the first two are pretty straightforward.

Closure under concatenation-- this is also going to be similar. You just try every possible way of cutting the string up into two pieces and run in parallel on. And if you ever find a way of cutting it up, and you run those two, and put those two sides in parallel, and if they both accept, then you can accept. And star is, again, very similar. So these are not too bad. But I admit, you know, it's not a whole lot of time to have to contend with something that you're just getting used to.

So let's talk about the very last topic of the day, which is really going to be setting ourselves up for Tuesday's lecture next week. And that's how we are going to be showing other languages are undecidable, which is something that I'm going to be expecting you guys to be able to do. This is the standard procedure for showing languages are undecidable using what's called the reducibility method.

And what that does is it takes, as a starting point, a language that we already know is undecidable-- typically, A,TM-- or it could be another one that you've previously shown to be undecidable-- and leverages that information to show other languages are undecidable. And it's using what's called reducibility. We're going to go into this more carefully next time. But basically, reducibility is a way of using one problem to solve another problem.

And so we are going to show, for example, let's take a look at the problem called the halting problem, which is like the famous problem for Turing machines. You just want to know whether it halts, not necessarily whether it accepts. So it's very similar, but not exactly the same.

And we're going to show that this halting problem is similarly undecidable. Now we could go back and do the whole diagonalization, but that seems like-- well, that's more work than necessary now that we already know A,TM is undecidable because we're going to show that we can reduce the A,TM problem to the halting problem. And we'll explain what that means again later. But the idea is-- and as we'll show in an illustration shortly-- that by proving by contradiction if the if HALT,TM were decidable, then A,TM would be decided. And we know A,TM is not decidable. And so that's our contradiction.

Now the way we're going to show that if HALT,TM is decidable, then A,TM is decidable is use a decider for HALT,TM to decide A,TM with a suitable modification. So basically, we want to turn a HALT,TM decider into an A,TM decider. And that's how we're going to reduce the problem of solving A,TM to the problem of solving HALT,TM.

Let's just do an example. If you've seen it before, obviously, this is not going to be hard. But for the many of you who have not seen it before, I'm partly doing it this time just so we can do it again next time. And maybe it'll sink in by virtue of repetition. So again, so as I just said, we're going to assume the HALT,TM problem is decidable and use that to show that A,TM is decidable, which we know is false. We showed it just earlier that it's not.

So assume we have a decider for HALT,TM. We'll call it R. And we're going to construct from R a decider for A,TM we'll call S. OK? So we're, again, typical proof by contradiction. We're assuming the opposite of what we're trying to prove. And then we're going to get something crazy.

OK, so here, my job now, is I'm assuming I have R, which is a HALT,TM decider. So now, I'm assuming I know how to decide if a Turing machine and an input eventually halts-- Not. Necessarily would it accept, just whether it halts. It's conceivable. You have to bear with me here.

It's conceivable that you could find a way to test whether Turing machines halt on their input, even though we now know that testing whether they accept their input is not decidable. So you have to be open-minded to the possibility that the HALT problem is decidable, and we're going to show that that's can't be. So we're going to show that if we could decide the halting problem, then we can use that to decide the acceptance problem. OK, so how are we going to do that? So imagine how we can solve the halting problem.

So to solve the A,TM, which is what my job is to do, so S is supposed to solve A,TM. I'm constructing a Turing machine as decide A,TM. I'm going to use first-- I'm giving it M and w. I'm going to feed it into R since that's really all I got. See if R tells me what happens. Does M on w at least halt? Well, if R says, no, it doesn't halt, well then I'm actually done because if M doesn't even halt on w, then it couldn't be accepting w. So at that point, I know that M doesn't accept w, and I can reject right off.

So R, you can see how it could potentially be helpful. But it's going to be helpful in either way because if R says M does hold, well, then, I'm also good because I don't know the answer yet, but what I do know is I can now simulate M on w until it halts because R has told me it halts. So I don't have to worry about getting into a loop. So S can be confident in being a decider for whatever it's doing because I'm running now M on w with a guarantee that it halts.

And now, that's going to tell me-- now eventually, the simulation of M on w is going to end up at an accept or reject. And that's going to be the answer I need. So if M is accepted, then accept. And if M is rejected, then reject. And that's how S solves A,TM using R, which solves HALT,TM. But S can't exist. And so therefore, R can't exist. And therefore, HALT,TM can't be decidable. OK?

So that quickly-- OK, I'm not sure which diagram you wanted me to show. But anyway, maybe we can do that. We're basically at the end of the hour or end of the 90 minutes. So let's do a quick review. And if you stick around, I'm happy to go back and look at any of the other slides that you might have missed something on.

OK, so just to recap, we showed that the natural numbers and the real numbers are not the same size using that definition of one-to-one correspondence to introduce the diagonalization method. We used the diagonalization method to show that A,TM is undecidable. We also showed that little theorem that if the language and its complement are recognizable, then the language is decidable. And from that, we concluded that A,TM complement is not recognizable. And then we showed, at least by virtue of an introduction to the method, the reducibility method to show that HALT,TM is undecidable.

And that was today's lecture. And we're and we're at the end of the hour. So why don't I-- we are finished. You can log out. And if you want, I will stick around.

OK, OK, this is kind of a good question here. So I'm getting a question about the A,TM-complement, which is-- since we have a recognizer for A,TM, if I'm doing justice to this question, we have a recogniser for A,TM, so why can't we just invert the answer? Flip the answer around, and now, we have a recognizer for the complement of A,TM, A,TM-complement. So why doesn't that work?

Well, the reason that doesn't work is because the recognizer for A,TM might be rejecting some things by looping. And now, if you just flip the accepting and rejecting, when it hits one of those halting states, it's going to give the reverse answer. But when it rejects by looping, it'll continue to reject by looping. So you won't get the complementary language coming out.

So if it would be helpful, I can go back to that slide here, which proves that A,TM-complement is unrecognizable because maybe we should start with the bottom. We know that A,TM is recognizable and undecidable, right? We already proved those two facts. A,TM is recognizable from the universal Turing machine, and it's undecidable by the diagonalization argument. Those two things together tell us that the complement has to be unrecognizable because if a language and its complement are both recognizable-- and we already know the language itself is recognizable. So now, if the complement is also recognizable, the language is going to be decidable by the upper theorem.

So it must be the case had either the language itself is unrecognizable, or its complement is unrecognizable. We know the language is recognizable. That's what the universal Turing machine told us. So the only thing left is for the complement to be unrecognizable. You should review that if you didn't get it because this is the kind of reasoning we're going to be building on things like that. So I think it's good to make sure you understand. OK.

OK, the diagram on the right-- so this is just a Venn diagram here. I threw this in at the last minute here. I was worried about it being confusing. That part is-- I'm trying to show that the three classes that we've already talked about-- the languages which are decidable, the languages which are Turing-recognizable, and the languages whose complements are Turing-recognizable. Those are three separate classes of languages. And those come up here in those three regions. These are the decidable ones. Here are the recognizable ones. And here are the ones whose complements are recognizable.

Now if a language is in both the recognizable, and its complement is recognizable-- so it's in both of these bigger regions here-- then this theorem tells you it's decidable. So that's why the intersection of these two regions is marked as being decidable because that means you're in both. OK? But we know that A,TM is sitting out here as recognizable but not decidable.

So A,TM is in the recognizable side, but it's not on the complement of recognizable, A,TM itself. The complement of A,TM is the complement of a recognizable but itself is not recognizable and not decidable. So you got this side of nice, you know-- I hope you think it's nice, but it's sort of a try to summarize things in this little Venn diagram.

So I think I'm going to then sign off. And I'll see you all on Tuesday. And have a good weekend. Bye bye.