[SQUEAKING]

[RUSTLING]

[CLICKING]

ERIK DEMAINE: All right, let's get started. Welcome back to 6.1200. Today we're going to talk about state machines, which is a very cool tool for thinking about processes over time. And because this is math for computer science, in particular we're thinking about analyzing algorithms that do a bunch of steps.

And the state of the machine changes over time, like the state of your memory changes. And state machines are a nice way to reason about how it's changing. In particular, we'll talk about an idea called invariance, which you use to prove algorithms and programs correct, or you will in the future.

And so today we're going to formalize that into a kind of induction, actually. You haven't seen the end of induction yet. So let me tell you a little bit about what state machines are. And then we'll prove some things and apply them to analyzing puzzles and other fun stuff.

So a state machine is going to consist of two things-- three things-- a set of states. And the idea-- this is just an arbitrary set. But the idea is that the state of your machine represents everything that could possibly be in the machine, all the information that's inside the machine.

We're going to start in some state, which we call q0. That's going to be an element of Q. And then we are allowed some transitions. And we're going to write a transition as q arrow r for some q and r that are states.

OK, let me draw you an example, very simple example of a state machine so we have some idea of what we're talking about. This is going to be an infinite counter. I'm going to draw it visually first. And then we can say what it means with all these sets. Just dot, dot, dot. Maybe I'll do one more.

OK, so this is a machine. You can think of it as a single register on your machine, single variable. It starts at 0. This little notation here means start state. There's an arrow coming from nowhere. So that's where you start.

You start in state called 0. So the set of states here is 0, 1, 2, 3, and so on, in other words, the natural numbers. And then when you're in state 0, the only transition you can do, the only arrow that goes out from there is from 0 to 1. And once you're in state 1, the only thing you can do is go from 1 to 2.

At some point, you decide to increment. Then you go from 2 to 3. At some point you decide to increment, and you go from 3 to 4, and so on. So this is a very simple state machine. It turns out it will be interesting for certain reasons.

But in general, you could imagine having all sorts of arrows. Maybe one has a transition to 3, for example. Or maybe 3 has a transition to 1. All of these are possible. You can have one state that has many transitions out of it. You can go in loops. You can do all sorts of crazy things.

But let me finish precisely specifying the counter. So we said here q0 equals the state 0. And what are the transitions? Transitions are all of the i to i plus 1 transitions where i is a natural number.

You may recall this set-builder notation from lecture 1, I think, where we write an element on the left, and we write some condition on the right. And so the idea is T is the set of all of these transitions i to i plus 1 for every i that's a natural number.

So that's an example. But in general, we can have all sorts of transitions from any state to any state. But you're only allowed to make the transitions that are in T. So let's formalize that with the idea of execution.

It's going to be a sequence of states that you can reach by a sequence of transitions. So let's write it as q0, arrow, q1, arrow, q2, and so on. Maybe up to qn.

And importantly, every transition we make for all i, the qi to qi plus 1, should be in T, not just any set of states here. We want to start in q0, our start state. We want to then make a valid transition from q0 to q1, then a valid transition from q1 to q2, and so on.

So in this example, the executions are very simple. In the regular counter without the red lines, the executions are just 0, 1, 2, 3. That's one execution. 0, 1, 2, 3, 4, 5, that's another execution.

There's an infinite execution, which is 0 all the way out to infinity. That's why I put this in parentheses. Either it stops at some qn, or it just-- or the dot, dot, dot continues off to infinity. So we're trying to simultaneously define finite executions, where you stop at qn, and infinite executions, where you go forever.

When you write an algorithm or a piece of code, you usually don't want your program to go forever. And so distinguishing between infinite and finite executions is one of the goals of this lecture. That will come at the end.

The other goal is to show that your program is correct in a certain sense, that in any execution, it does the things that you want it to do. So that's what we're going to formalize in a moment. But let me first talk about reachable states, the idea of reachability.

I'm going to call a state r reachable if there is an execution that starts at the start state q0 and ends at r. So in the counter example, every state can be reached. If I pick a state i, I can go 0, 1, 2, 3, up to i.

But if I drew some other circle over here, like "bad," with no arrows coming into it, you can't reach "bad." So usually we're only interested in the reachable states because you can't get anywhere else. I feel like I should make a *Reacher* joke, but I can't think of one.

I think, at this point, I want to talk about a more interesting example. And then we will come back to the theory. So the interesting example is this. It's called the 8 puzzle. You have a 3-by-3 grid with one hole and 1-by-1 pieces in it. And you can slide them around.

And so this is the solved state. Your goal is to get here. But you start in some random state. And you want to move the pieces around to get there. So what we'd like to do is capture how to solve this puzzle or actually, maybe more interestingly, is when it's solvable.

So here, for example, I'm almost solved 1, 2, 3, 4, 6, 5, 7, 8. I wanted to get it to 1, 2, 3, 4, 5, 6, 7, 8. It turns out that puzzle is not solvable. I claim it's impossible to get from here to the goal. So let's talk about the 8 puzzle.

So I have some examples here. 1, 3, 4, 2, 5, 7, 8, 6. Going to 1, 2, 3, 4, 5, 6, 7, 8. Or this one-- 1, 2, 3, 4, 6, 5, 7, 8. Hopefully the rules are clear. We'll formalize them in a second.

I claim it is possible to go from this arrangement to this arrangement. It's actually quite easy. You move the 3 over. You move the 2 up. You move the 5 over. You move the 6 up.

So this is possible. I claim this one is impossible. And one of the goals-- our first goal really is to prove that, that you cannot swap the 5 and the 6 by themselves. We're going to build up tools to do that.

So the counter was a very simple state machine. Let me now define a more interesting state machine, which is the 8 puzzle. Remember, the set of states q is supposed to encapsulate everything about the puzzle or your system or whatever.

So this is going to be all the different arrangements of the eight pieces, plus a blank, in a 3-by-3 grid. So I'm just going to write that in words. All possible arrangements of eight-- I'm going to call them of 1, 2, 3, up to 8, and the blank, in a 3-by-3.

In later parts of the class, we'll be able to count how many of these things there are. It's nine factorial. But we don't really care how many there are right now. We just want to think about this as a set of all the possible states of this board.

Our start state, well, that's either this one or this one, depending on where we want to start. We're also going to define a final state, q sub f. And that is sorted by reading order.

So this is q sub f. Reading order-- like in English. You read the top row, second row, third row, with the x at the end. That's always our goal in this puzzle. That's our definition.

And then, what about the transitions? Transitions are going to look like some q to r for two states q and r. And it's when you can get from q to r by making one slide.

I'm just going to write that in words-- can get from state q, that 3-by-3 grid, to state r in one slide, meaning-- meaning you exchange the blank, the x, with one of its neighboring cells. So in particular, there is at most four different transitions from any state.

You could move the thing above the blank into the blank. You can move the left, right, or below the blank into the blank. But if it's on the boundary, you have fewer than four moves.

So hopefully now I've said it enough times the rules are clear if you haven't played this game before. 15 puzzle is much more common, but 8 puzzle is a little easier to think about-- smaller pictures, easier to draw, and I can actually solve it in a reasonable amount of time.

OK, so this is a claim that we want to prove. You cannot get from this state, with 5 and 6 flipped, into this state. And so now we're going to build some technology to do that. And that technology is a-- first part of it is called a state predicate.

So we've seen predicates before. Predicate is just a-- a proposition that depends on some variable. And here, the variable is a state.

So P of q is either true or false depending on which state you are looking at. You can also think of this, if you're comfortable with functions-- we'll talk more about functions later in the class-- this is a function from q to the Booleans true or false. It's an equivalent way to think about it, whatever you're more comfortable with. For every state, this predicate assigns true or false.

Now we're going to define two types of predicates that are particularly interesting. A predicate is preserved if whenever there's a transition from q to r, P of q implies P of r.

What does that mean? It means that if I start in some state where P is true and I do a transition, then P remains true. So the truth of p is preserved along transitions. We'll make this concrete in a second.

Second important notion is called an invariant, or invariant state predicate, if P of q is true for all reachable states q. Crucially, in this definition, we're only interested in the reachable states.

That's kind of important for the 8 puzzle because we claim some states are unreachable. Like, if I start here, the claim is that this one is unreachable. That's what we'd like to prove actually. So this seems useful.

In particular-- let's write this fun fact-- if P of q is invariant, meaning it's true in all reachable states, and P of q is false for, maybe I'll write P of r here, then r is unreachable. That's pretty much by definition and maybe the contrapositive.

This is saying invariant means that for all reachable states, P of q must be true. So if I can find a particular state r where P is false, that means you cannot reach it. And that's what we'd like to show. We'd like to show this state is unreachable from this one. So maybe we can use these state predicates to do that, and indeed we can.

Let me tell you another tool. How do we prove that a state predicate is invariant? This seems a little bit tricky because this is talking about all reachable states. Reachable states is huge. I don't know.

Preserved seems relatively easy to prove. This is just saying, for each transition, something implies something. So I'd like to prove preservedness. And I'd like to use that to get invariant. And we're going to do that using induction, but a particular form of induction called the invariant principle.

So this is for a state predicate P of q, if P of q0 is true and P of q is invariant-- get it right; preserved-- then P of q is invariant. This is what I said I wanted.

On the one hand, invariance seemed hard to argue. Preserved seemed easy to argue. And this is the theorem that connects them in the way that we want. It says, if you can prove preserved and you can prove that P is true in the very initial state, then it must be invariant.

I think this is intuitive, right? If you start true and preserve says, every time you do a transition, you stay true, then you're always going to be true in any reachable state, anywhere you can get. In fact, this is induction in another form.

So if you recall in the past, which I happen to have right here, induction, we have-- regular induction says for predicate P of n over natural numbers instead of states, if P of 0 is true, and for every natural number P of n implies P of n plus 1 is true, then for every natural number P of n is true.

This is saying the same thing. Actually, it's exactly the same thing if you apply this invariant principle to this counter. In the counter, the states are the natural numbers. And the transitions are exactly going from i to i plus 1.

So the invariant principle is actually a generalization of the induction axiom. The special case here, where we plug in the counter as the state machine, this is true for all state machines. And if we plug in the counter for our state machine, we get exactly the induction axiom.

This is our base case in some sense. We prove that P is true in the initial state. This is our induction step. It says that if we start at q and we transition to some other r that's reachable via transition, and in the particular case of a counter that is going from q to q plus 1, then we get that P of q is true for all reachable states. And in the counter, everything is reachable.

So invariant principle implies induction axiom. Conversely, we can prove the invariant principle using induction. Maybe it's useful to see that proof.

So let's prove the invariant principle by induction. So it's not so obvious how exactly. But the idea is to specify a particular induction hypothesis, which is, for all length n executions, P of qn is true. There should be a colon out here.

I think it's intuitive that this is enough. But we'll justify why this is enough. The idea is if you have some state that's reachable, it's reachable by some finite execution of some length. And we're going to prove for all n that all length n executions P of qn is true.

And by inducting over length, we essentially start with short executions and then make them gradually longer. And that makes this easy to prove. So let's do a base case, IH of 0. Well, that's all length 0 executions. There's only one length 0 execution.

That is just q0. You start at q0, and you stay there. That's length 0. And we claim that P of q0-- q0 is clearly a reachable state. You get there via length 0 execution. Why is it true? This. It's what we assumed.

OK, so by 1. So that's the base case. Of course, the more interesting one is the induction step. I'm going to assume-- we get to choose.

We could follow this plan, assume P of n And prove P of n plus 1. I find it slightly more intuitive to assume P of n-- or sorry, instead of P, it's IH here for induction hypothesis-- IH of n minus 1. And we're going to prove IH of n.

What does it mean to prove IH of n? It says, well, for all length n executions, P of qn is true. So how do we prove it for all? We just take an arbitrary thing in the quantifier. So take any-- consider any length and execution.

And then we need to prove the property we want, which is P of qn. So let me give some notation for that execution. Any length n execution q0, q1, qn minus 1, qn.

And what we want to show is that P of qn. That's our goal right. It says, for all length n executions, P of qn is true. So we take an arbitrary execution. If we can prove P of qn is true, then we're done.

Well, somehow we want to use induction here. We've assumed the induction hypothesis for size n minus 1. Can you see a length n-minus-1 execution here that might be interesting? Yeah?

AUDIENCE: q1 to qn.

ERIK DEMAINE: q1 to qn. That's a good idea. Does that work? Is that an execution? Let's go back and check. Execution of a state machine is an infinite or a finite sequence starting with q0 and maybe ending. But it needs to start with q0.

So there's two natural choices here. One would be starting at q1 and going to qn. But q1 might not equal q0. So that might not be a valid execution. So we wouldn't be able to apply induction to that.

The other obvious choice is q0 to qn minus 1. That is a valid execution of length 1 smaller, length n minus 1, because it starts with q0. A lot of terms here. So it's easy to lose track. But eventually, they will become comfortable.

So by induction hypothesis of n minus 1, if we apply it to this execution of length n minus 1, we learn that P of q n minus 1 is true. That's what the induction hypothesis says. The last state in your execution is true.

So now we know P of this is true. We just need to prove P of this is true. Why is that? Because property 2-- P of q is preserved, which we can unroll over here. Says, for every transition, in particular the one from qn minus 1 to qn, P of q minus 1 implies P of qn.

So by 2, P of q n minus 1 implies P of qn because this is a transition in T. So P of qn. So that's the end of the induction step.

So by induction, we get, for all n, induction hypothesis of n. And the idea is that this implies P of q for all reachable states q because if you have a reachable state q, there's, by definition, a reachable way over on the left. There's some execution that gets there, and it has some length. And so IH of that length gives you what you want.

So that's how we prove the invariant principle. It's a pretty intuitive notion. But may be nice to see yet another example of an induction proof. Great. So how do we use this?

Let's go over here. Here is a template for proving a theorem, which is P of q is invariant. This is the thing I keep saying is hard to prove. Or by the definition, it seems hard to prove. But using this invariant principle, it's going to be easy. Here's what a general proof will look like.

We can say, proof by invariant principle. For the invariant principle to apply, we need to prove two things-- number 1 and number 2. So number one is prove that P of q0 is true. And in a template, we just say because reasons which we don't know, or we have to fill in.

And the second thing is that P of q is preserved. That's the more interesting thing. And for this, we need to unroll the definition of "preserved." Says for all q, all transitions q to r, P of q implies q of r.

So how do we prove that? We just take any transition in t, q to r. And because it's an implication, we can assume the left-hand-- if we want to prove an implication, we can assume the left part-- this is the direct proof-- and prove the right part.

So we can assume P of q is true. And then what we need to do is show that P of r is true. That's the interesting step, which we need to fill in.

But this is the template we're going to now use to analyze the 8 puzzle. Of course, I haven't defined what predicate we're analyzing. That's the next step.

But once we have a predicate, we'll be able to prove that it's invariant using this simple structure. Just check it. Check that it's initially true. And check that if you start from something that's true and you take a transition out of it, it's still true. Great.

So let me define a predicate for the 8 puzzle. Maybe I want it closer to these examples, actually. Yeah, let's go here. I think you know the definition of a state machine.

OK, so for this 8-puzzle state machine, I'd like to define a bit of a bizarre predicate. So first, I'm going to define the notion of an inverted pair.

This is going to depend on what state we're in. So let's look at inverted pairs in this example and use three examples. So an inverted pair is two numbers, so not the blank but two numbers between 1 and 8. And I want i less than j. And yet I want j to appear before i in the reading order.

So for example-- my red chalk-- in this picture, this state, 2 and 4 are inverted. In reading order, 4 comes before 2, yet 4 is greater than 2. Any others?

8 and 6 are inverted. 6 and 7 are inverted. 7 is bigger than 6. 8 is bigger than 6. But they come earlier than 6 in the reading order. I think there's one more. 3 and--

AUDIENCE: 2.

ERIK DEMAINE: 2. Thank you. That's one's a little trickier because it's on different rows. But in reading order, 3 comes before 2, and yet 3 is greater than 2. So there are four inverted pairs in this example.

How many inverted pairs are there in this example? Zero. Zero inverted pairs. How about this one? One, yeah. I thought you were holding up your hand. But you're holding up one finger. [CHUCKLES]

6 comes after 5. That was the only thing out of order. That's what we started out with. Where do you come up with this idea? It just is fundamental. You're supposed to magically invent it. But if you fool around with the puzzle enough, it's actually somewhat intuitive in that you can get where it's almost sorted, but there's just one pair that's wrong.

So you might think, well, maybe there's something about these inverted pairs. Let's talk about them in general, not just about 0-versus-1 inverted pairs. But let's count them.

So over here, we had one inverted pair. Here we had zero inverted pairs. Here we had four inverted pairs. And I claim it was possible to go from four inverted pairs to zero but impossible to go from one inverted pair to zero. Why? Parity, it turns out.

"Parity" means odd or even. So P of q is going to be that the number of inverted pairs in state q is odd, turns out to be. This proof will work for either one. But if we want to prove this particular impossibility result, odd is what we need.

So p of q is going to be true for this configuration because it has an odd number of inverted pairs. And it's going to be false for these two. Why is that interesting? Because over here, we said, suppose we can show that P of q is invariant. I claim that's true.

If we start from here, where P of q starts true, because the number of inverted pairs starts odd, and we find another state where P of r is false, because here the number of inverted pairs is even instead of odd, then r is unreachable. That means it's impossible to get from this state to this state. That's what we want to prove.

So if we can prove this invariant property, we can plug it into here and get the puzzle impossibility result that we want. That's where everything's going to come together. So let's prove that P of q is-- ideally, we'd like it to be invariant. But to prove invariant, we're going to start with proving that it's preserved.

The reason for the 8 puzzle, that I want to talk about preserved first, is to be invariant, it depends where you start. And that's a little arbitrary. There isn't one natural start state for the 8 puzzle. You might start here. You might start here. You might start here. That'd be a very easy puzzle.

But preserved is just a general property about the system. So I don't need to assume anything for that. So we want to do this part. We want to prove P of q is preserved. So we consider any transition q to r where P of q is true. We need to prove P of r is true.

So here, let me plug in the outline of step 2 there. Consider any transition q to r that's valid, it's in T, where P of q is true. And we want to show P of r is true.

What do transitions look like in this game? Well, relative to reading order, reading order talks about rows. So there's really two types of moves we care about. There's horizontal moves left and right. And there's vertical moves up and down. I claim those are the two cases that we need to think about because we're thinking about reading order.

So case 1 is that this transition, q to r, is a horizontal move. Horizontal move. So this is like we take some number i, and next to it there's an x.

And maybe if we move to the right, that would be x i. Or if we move to the left, it would be like that. It's one of those two transitions. x here means the blank. How does that change reading order? Yeah?

AUDIENCE: Nothing.

ERIK DEMAINE: Nothing. Nothing changes. The reading order is the same. When we're talking about reading order, we're only talking about the numbers. We don't care about the x. So no changes in reading order or in the number of inverted pairs because that's only dependent on reading order.

So we have P of r. Case 1 is easy. Let's prove case 2 maybe over here. Case 2 is a vertical move here. There's actually something to show-- more interesting, but not much harder.

OK, I'm going to draw some pictures for intuition. It could be we are moving i down, and there's two things to the right of i. It could be we're moving i in the middle, to here. Or it could be we're moving i on the right, down to here.

And this is for a down move. Up is going to be symmetric. But let's first think about down moves. So in order to do a down move, in reading order, there are two items that come after i. Because we're moving down, there is something after i. There's a whole row after i until this x.

So if we can move i down, there's two numbers between i and the x in reading order. And i starts before those numbers if we're going down and ends after. So the move here, we can think of it as swapping-- we want to swap i with x.

But this is equivalent to swapping i with j, first of all, and then swapping i with k. Instead of going straight down-- and another color-- I'm going to think of swapping i to first come after j and then come after k. That's the same as the x.

And it doesn't matter where we started. It could be here, and then we go here. Or it could be here, we go after j, and then we go after k. The point is if we swap-- if we figure out how to swap two adjacent numbers in reading order and we swap i twice, then i gets to where the x is because this is a 3-by-3. That's why twice.

So now what we care about is what happens when we swap two adjacent items in reading order, so not adjacent in the 3-by-3 grid but adjacent in reading order. That's what we care about. If we can solve this and we do it twice, then we will know what happens when we do a vertical move.

So what happens when I swap two numbers i and j? How does that change the reading order? I mean, it swaps i and j. That's all. Everything before i is still before i. Everything after j is still after j.

The only thing is we're changing-- if they're adjacent items i and j, we're only flipping whether ij is inverted. If ij was in the correct order before, if i was less than j, then, after I do the swap, they're an inverted pair. If they were an inverted pair initially, if i is bigger than j, after I do the swap, they're no longer an inverted pair.

So swapping two adjacent items changes the number of inverted pairs by plus or minus 1. If they were inverted before, it decreases the number. If they weren't inverted before, it increases the number, but always by one. Never stays the same. Never changes by seven.

So when we swap two adjacent pairs, we change by plus or minus 1. When we do a vertical move, we do we want to do two of these adjacent swaps. So let's continue over here.

So if we do two adjacent swaps, this will change the number of inverted pairs by plus or minus 1, plus plus or minus 1. Were each of these pluses or minuses? We don't know. They could be plus or minus.

But the point is plus or minus 1 is an odd number. Plus or minus 1 is an odd number. When I add two odd numbers, this is an even number.

Turns out you can draw a little picture. If we start at 0 here and we do plus or minus 1, and then from here we do plus or minus 1, and from here we do plus or minus 1-- this is a case analysis of all the things you could do-- you either get 2 or 0 or minus 2. These three numbers are all even.

But I don't particularly care. It happens to be just these three values. What I care is that this number is even. We started with P of q being true over here. Assume that P of q is true.

That means that the number of inverted pairs in state q is odd. And now we change that number by an even amount. Therefore, it will be odd still. If we take an odd number, we add an even number, it stays odd. Therefore, P of r is true. Question?

AUDIENCE: What happens if you swap-- if you swapped the i and the j?

ERIK DEMAINE: So the question is, why is this two swaps? So the funny thing here-- so it's right. If I swap i and j, then-- let me maybe draw a picture somewhere here. Let's just go here. Scratch space.

So if we have i, j, k, x, and I swap i and j, then I get j, i, k, x. And then I swap i and k. Then I get j, k, i, x. And you're right, i is not quite where I want it.

[CHUCKLES] This is funny. I really want i to be here, which I can do like this because that doesn't change reading order. OK, it's good that we spelled this out.

And also, I can do horizontal moves for free. So j, k, i. This is maybe a way to think about it. The point is that these three have the same reading order. So those don't really count. But I'm glad you had me spell it out.

AUDIENCE: So the [INAUDIBLE] we're doing right now are not part of the valid transition. It's just to just separate them?

ERIK DEMAINE: Right. These are not valid transitions, indeed. At the end, we get to something that's a valid transition. So the transition goes from here directly to here. And we're splitting it up into some sub moves that aren't valid moves by themselves but easier to analyze.

First, we think about what happens when we swap i and j. That changes by plus or minus 1. Then we swap i and k. That changes these by a plus or minus 1.

Then we do these horizontal moves, which doesn't change anything, or horizontal moves. Moving the x in reading order doesn't change anything. Cool? All right, great. I feel like I skipped a little bit of a step here.

So the point is, when we swap-- in case 2, when we do a vertical move, we started with an odd number of inverted pairs. And then we add an even number. And odd plus even equals odd. Therefore, P of r is true. That's the point.

So if we start odd, we will stay odd. If you swap odd and even in this proof, you can also show that if you start even, you stay even. But what we really want is, for this part here, we want to get P of q to be invariant and find a place where P is false, and then conclude that r is unreachable. So let's do that quickly.

So let's define this state to be q0. So for this q0, we have-- or maybe let's say if P of q0 is true, as in this example because there's an odd number of inverted pairs, then P of q is invariant.

And so by this property, if P of r is false, then r is unreachable. So actually what I care about is not p of r but P of the final state, qf. So that's exactly the scenario.

We start in a state where we have an odd number of transitions. We're trying to get to a state with zero inverted pairs. And so that's not going to be possible because we start with P true. And we somehow are trying to reach a state where P is false.

But if we can prove this invariant property-- and we've basically done all the hard work here. This is just by the invariant principle. So now I'm following the outline over on the left. Step 1 is-- or part 1 is that P of q0 is true. We assume that.

Step 2 is that P of q is preserved. And we just proved that. This is our theorem. So maybe you don't know this word, "corollary"-- or "cor-aw-luh-ree" if you're Canadian. "Corollary" means something that follows.

So "theorem" means something that's true. Both of these are supposed to be true. But these are words to say that a theorem was the main thing we wanted to prove. And then corollary is something that follows immediately from it. It's almost a one-line proof here.

Once we have the theorem, the corollary is immediate. It's just some terminology. So hopefully you now believe this is impossible. Any questions about the 8 puzzle?

For fun, let's solve it one more time. I actually didn't have time this morning to implement the "only generate solvable puzzles." So I have no idea whether this will be solvable. It's kind of exciting. If you can count the inversions, you'll figure it out. But I keep moving it, so it's pretty hard.

And I'm following a particularly slow algorithm for solving this puzzle, which I won't describe. But it's on wikiHow if you're interested. Oh, I solved it. Great.

So I must have started with an even number of inverted pairs. In a future version, I will add that number. But there we go. Cool. Question?

AUDIENCE: [INAUDIBLE] final stage [INAUDIBLE]?

ERIK DEMAINE: So your question is, if, in this state, we move six down, how many inverted pairs do we get? Let's draw it. So that is 1, 2, 3, 4, 5, x. It should be 2. 7, 8, 6. How many inverted pairs are there? 7 is inverted with 6. 8 is inverted with 6. And that's it.

So there's an example in action. Vertical move, changing by 2. Sometimes it changes by 0. If i is sandwiched in between j and k numerically, sometimes it changes by plus or minus 2. Of course, if we do the up move, we subtract 2. There was another question. Yeah?

AUDIENCE: Can you explain the definition of preservation and invariance? And [INAUDIBLE]?

ERIK DEMAINE: OK, question is explain preserved and invariant. And where are we using the different parts? If I can make an induction analogy for a second, preserved means-- is like your induction step. It says, every time you do a transition, if P started true, it remains true. And invariant is what you get out of the induction.

Let's go over to the invariant principle here. The invariant principle basically says preserved implies invariant, except you also need to check that you start true. If P starts true and whenever you do a transition it remains true, then it's always true for every reachable state.

OK, so crucially in this setup, we're ignoring all the unreachable states, just talking-- invariant only talks about reachable states. It says that every reachable state P is true. That's what we're concluding from, we start true, and every time we start in a true state and we make a transition, we remain true.

Hopefully that's clear. Takes a little while to get comfortable with these definitions, but very important, the difference between preserved and invariant. So if you forget, read the definition, or just follow this template. This makes it pretty clear.

Usually, we just want to prove invariant. And there's a clear start state. So in fact, the next example will have a-- well, no, that's not true. But if there's a clear start state, you can just follow this template. And all you care about is proving invariant.

OK, let's do the next topic, which is termination. And maybe I want to start with an example, which is a simple sort of machine. This is another state machine. And it sorts n numbers.

This is a problem you study a lot in the follow-ons to this class, the algorithms classes. So suppose we're given a bunch of real numbers, or integers, whatever. We want to sort them in increasing order.

OK, here is-- so they're given in some particular order, a1 through an. And I want to rearrange them so it has exactly the same set of numbers-- or exactly the same numbers but in a different order, and this time increasing order. So I want a1 to be less than or equal to a2, less than or equal to a3, and so on.

Here is a very simple algorithm to do that, which is while there's some index i, where ai is bigger than ai plus 1, swap ai with ai plus 1. So I'm just looking at consecutive pairs ai to ai plus 1. And if there's some that's out of order, i is less than i plus 1 but ai is greater than ai plus 1, then I want to swap them in the right order.

We can see this algorithm in action for fun. And then we're going to convert this algorithm into a state machine. So just pretend you understand algorithms for a second.

Here, we have the numbers in a particular order. The height corresponds to the value that's written, 3, 6, 2, 5, 4, 1. And we're just going to do an example. But then we do a swap.

A swap-- we're always swapping adjacent pairs. And then Pac-Man comes over to the left, and we get them in sorted order. So that is the algorithm. Let's now analyze it as a state machine.

So first question is, what are the states Q? Well, throughout this algorithm-- so this is in our head, what we're thinking about doing. And we choose Ii arbitrarily. Now we're going to model it as a state machine. This will make precise what's actually going on.

As we do this, we're taking the original items in some order. We're always swapping items. So at any moment in time, what we have is some reordering of the ai's. Those are our states.

Let's define Q to be all possible orderings. These may not all be reachable, but this is the thing that's easy to define-- all possible orderings of a1 to an. For this template-- OK, what are the transitions?

Transitions are going to take some ai ai plus one pair and swap them. So if we start with them in this order, they're going to end up in the opposite order. And we are only allowed to do this when ai is bigger than ai plus 1.

So that's the transitions. Implicitly, this specifies exactly the algorithm. The algorithm says, take some i where ai is bigger than ai plus 1, and swap them. That's exactly what the transitions specify because we can take any transition that's valid from any state.

So if we can find any i where ai is greater than ai plus 1, then swapping that pair is a valid transition. We don't know which one will do. State machine will follow some execution which will do one of these until it can't anymore.

OK, great. Can't anymore. Let's define a tool which is final states. Let's call a state q final if there's no transition q to r and T, if there's no transition that starts at q and goes somewhere.

So in the 8 puzzle, there are no final states. You can always make moves. You never are done. You're done only when you happen to reach the final state. But there's nothing in the move definition that prevents you from continuing.

In this problem, there are final states. I claim it's exactly a0 to an, where a1 is less than or equal to a2, is less than or equal to a3, and so on, is less than or equal to an, namely the sorted order. Why?

Well, first of all, I claim this state is final because if you look at any pair ai, ai plus 1, we have ai is less than or equal to ai plus 1. So we don't have a transition out of there. So these states are definitely final.

I also claim that it's all the final states because if I had some state where this is not true, that means one of these inequalities is not true. So that means there's some ai that's greater than ai plus 1. And then there's a transition.

So that was a spoken argument for why the final states are exactly this. Technically, we're proving all of these states are final and that all final states are of this form, and therefore these are all the final states. OK, cool.

So this gives us something we call partially correct. I'm not going to define this formally. Partially correct machine is a machine where, if it terminates, which I haven't defined, then gives the correct answer.

What does "terminate" mean? OK, to terminate means-- when we're going to say a state machine terminates, really what we're meaning is it always terminates. No matter how you execute this machine, it stops. And the only way it can stop is that it reaches a final state.

If you think about it for a second, the definition of "execution" was a sequence of transitions. And so terminates says there's no infinite execution. For an infinite execution, that means every state qi, there's some transition out of it to qi plus 1. So if that's not possible, that means that, at some point, you reach a state that's final. And from there, there's no transition.

So you could either say "it terminates" means there's no infinite execution or every execution reaches a final state. These turn out to be equivalent. So now I claim what we've shown is that, for the simple sorter machine, it actually sorts in this partially correct sense that if the machine terminates, then it will give the correct answer because if it reaches a final state, it's sorted. That's what we just argued.

Now, that's not enough. We also want to show that it terminates. And so really, the main thing to do-- the main thing to do-- is to prove termination. And in particular, let's say a machine is correct if it terminates and produces the right answer, which was partially correct.

OK, I haven't formally defined "right answer." But for the sorting problem, it's clear. We want to get them in order so that this holds, a1 is less than or equal to a2, is less than or equal to dot, dot, dot.

OK, so we've proved partially correct. But we need to prove correct. And in particular, we need to show that it terminates. So we need some tools for proving termination for this machine.

Let's go to another board. We want to make sure that this algorithm doesn't take forever, it doesn't go in an infinite loop and never reaches the sorted state. Let me define a general tool for this. This is how we will always prove termination in this class.

It's called a derived variable. This is a reference to random variables, which we will see when we get to the probability section of the class. Lots of forward references today.

It's very simple. It's just, for every state, we define a real number, x of q, however you want. In other words, this is a function from the set of states to the real numbers. In our applications, we're only going to think of the natural numbers here. But in general, this could be a real number.

And we're going to call it a derived variable strictly decreasing if whenever we have a transition from q to r in T-- it's a valid transition-- we have x of q is strictly greater than x of r. The new value is strictly less than the old value. That's strictly decreasing.

This is a very strange looking colon. So again, this is about a state machine. So we can talk about transitions. And every transition strictly decreases the derived variable.

This is going to be our analog to state predicates and preserved invariance. State predicates we map to true and false, or 0 and 1. Now we're generalizing to allow it to map to any number, any real number. But very similar notion. Just we're outside the Boolean realm now.

It's very odd to talk about strictly decreasing for Booleans. We need this. OK, here is a wonderful theorem. It'll be obvious once I state it. But it's very useful for proving termination. It's how most termination arguments go.

So it says, suppose you have a derived variable x of q. And suppose it satisfies two properties. Property 1 is it's actually over the natural numbers, not reals.

And property 2 is that it's strictly decreasing. Then, state machine terminates. So the "then" is, of course, what we want. We want to show that our state machine terminates.

So I claim this is all you need to show. You need to construct some derived variable where the value is always a natural number. So crucially, it's always non-negative. And it's always an integer.

I highlight this instead of putting it in the definition because this is something we really need to prove. This theorem is false if you allow real numbers. It's false if you allow negative numbers. Very important that they're non-negative integers.

So that's something we need to check. And it's strictly decreasing. That's all we need. Then we're guaranteed the machine will terminate. Why? Any guesses why? Why is this guaranteed that our machine terminates? Yeah?

AUDIENCE: Well, when you reach 0, if x sub q is strictly decreasing, there's nowhere to decrease. So there must be no more transitions.

ERIK DEMAINE: That sounds good. So when x of q reaches 0-- I don't know if it does, but if it reaches 0, you definitely can't go any farther because there's no transitions out of 0. You can't strictly decrease from 0 if you're non-negative. Why do we get to zero? Yeah?

AUDIENCE: We don't necessarily. But based on the construction of natural numbers, we will reach a minimum because it's bounded below, [INAUDIBLE].

ERIK DEMAINE: OK, we don't necessarily reach 0, but we get to some minimum. Yeah, I think these are all in the right direction. I'll write down what I would call a sketch that's thinking about it kind of the other way around.

Instead of focusing on x of q decreasing, I want to think about executions of this machine. We want to prove that the machine terminates. That means that there are no infinite sequences. In other words, all executions have finite length.

I claim all executions have length at most x of q0 because every time I take a step, every time I do a transition, x decreases by at least one. If it starts at x of q0, it can only decrease by 1 that many times. And at that point, it will have to have reached 0, and then it has to stop.

Maybe it stopped earlier. But this guarantees that it stops. Every step I make, I strictly decrease x of q by at least one. We start here. We can't go below 0. So the number of moves is at most that.

You can prove this by induction. But it's kind of straightforward. So you're in the right idea. But this is a little more precise. Great.

So now let's apply our termination theorem to the simple sorter machine. But to do that, we need to define one of these derived variables. And-- [CHUCKLES] --it's actually this one. I'll be mean and just erase this. There's our derived variable. I'll rewrite it.

So there's a reason these two examples are in the same lecture. They use the same idea, which is inverted pairs. Oh, perhaps I shouldn't have erased the definition of inverted pairs. Sorry.

So back to the simple sorting machine. We're going to define P of q to be the number of inverted pairs in state q. That's some number. It's a natural number. It's always greater than or equal to 0. It's always an integer because inverted pairs were this discrete object.

It was a set ij-- oh, maybe I need to redefine this a little bit. It's no longer reading order. So what is an inverted pair? This is ij, where i is less than j-- sorry-- but ai is greater than aj.

So i and j here are indices into this sequence. So in this example, let's say 6, 2 is inverted because 6 is to the left of 2. But 6 is greater than 2. And that's the opposite of what we want.

But also, 6, 1 is inverted. It's not just adjacent pairs-- all the pairs. So there's a bunch of pairs here. How many? It's exactly the number of dots in this diagram.

Each dot-- which we construct by taking each of these numbers of the particular height and drawing a leftward ray, a line to the left. And every time we hit another number, that means that number was taller than I was.

So 1, for example, is wrong with everybody because it's supposed to be all the way on the left. So with this funny height diagram, you just draw these lines. Those are the inverted pairs-- a nice visualization. Thanks, Zach.

TA: And Brynmore.

ERIK DEMAINE: And Brynmore. So now we think about what happens when we do a swap. So for example, here I swap these two items. Oh, great. Before, 6 and 2 were inverted; now they're not.

And that's all that changed because in the algorithm, in the transitions, we're only swapping adjacent pairs. If I swap it back, the number increases. And now if we watch this algorithm, each time it swaps an adjacent pair, it decreases by 1, the number of inverted pairs. Watch it one more time for fun. I love this animation.

I claim this P of q is strictly decreasing. Why? Because every time I do a transition from q to r, P of r is 1 smaller than P of q. Let's look at the definition of transition over here.

When I do a transition in this algorithm, I only swap two adjacent elements. So the inverted pairs with everything else doesn't change. And these guys were guaranteed inverted. I told you that ai was bigger than ai plus 1 whenever I did a transition.

So I decreased by 1 the number of inverted pairs here. So I strictly decrease by exactly 1 every single time. So in fact, the running time of this algorithm, the number of swaps that we do is exactly the number of inverted pairs.

But in particular, it terminates in finite time. That's what we get out of this termination theorem because we checked this function. This derived variable is non-negative. It's an integer. And it's strictly decreasing. Therefore, our sorting algorithm actually terminates. Yay.

I was supposed to do an announcement about problem sets--

TA: Yes.

ERIK DEMAINE: --which was-- the-- oh, yes. We're about to release solutions to a problem set-- problem set 1. And you may recall you can submit your problem set 1 anytime between now and the end of semester. So what do you do? Are you allowed to look at those solutions? Yes, you're allowed to look at the solutions.

But you cannot copy those solutions. You can look at them, then put them aside, remember the solution, think about the solution, write it in your own words, how that solution works, and put it at the top of your document-- it says in the collaboration policy-- "I looked at the solutions, but I am not copying them."

If you copy them, you will be penalized, OK? Just understand how the proof works, and then write down how the proof works in your own words. See you next time.