PROFESSOR: Just as truth tables provide a simple, direct way to define the meanings of the individual propositional connectives or propositional operators, they also provide a methodical way to understand the behavior of formulas, and in particular whether two formulas are equivalent or whether a formula is valid, meaning that it's always true. So let's take a look at that.

To being with, if I'm thinking about a propositional formula that's a composite one that's built out of lots of more atomic, primitive ones, then in order to figure out the value of the whole thing, I need to know the values of the individual components. So if we think of a formula involving P's and Q's and R's that are true/false value propositional variables, then I need a truth assignment to know the values of those variables, in order to know whether or not the formula is true or false.

So in computer science jargon, this process of assigning values to variables is called an environment, although logicians would call it a truth assignment. So an environment tells you, given a variable, whether or not it's true or false.

Let's look at an example of three variables-- P, Q, and R. They are true/false valued, and I'm going to tell you that I've got an environment v in which P is true and Q is true and R is false. So v of P was T, et cetera. So I'm thinking of v as a function that maps a variable to its value.

Now, let's see how I would use this particular environment to figure out the value of this composite formula whose atomic parts are P and Q and R, and Q again, I guess. Let's take a look at how we would go about figuring out the value of this whole formula given the values of P, Q, and R. It's pretty straightforward, but the methodical way to do it is sort of from inside out.

Let's begin by attaching the truth values that we're given to those particular variables. Now, notice here that I've got both arms of the AND have been assigned truth values, and they're both true. That means I can assign true to the conjunction formula, the AND formula, and I do that by putting the T under that AND, which is the principal connective of this subformula.

Now, looking back at this Q, I've got it's true, which means that NOT of Q is false, so I can put a false under the NOT. Now, notice I've got both arms of this XOR are defined. They're both false, and that means that the XOR is false because it's only supposed to be true if exactly one of them is true.

And next, I had this true for AND. That means that NOT of that formula is false. And now that I have both arms of the OR, this one is false and that one is false, I can conclude that the entire expression is false by putting that final truth value under the principal connective of the whole formula.

So that's actually an easily defined recursive process. I've described it from inside out, but if you are programming it, you would be doing it recursively top-down in order to evaluate the truth value of a formula given the truth values of its constituent variables given an environment.

Now, a basic idea about propositional formulas is that two of them are equivalent if and only if they have the same truth values in all environments-- in all environments. No matter what the values of the P's and Q's and R's are, no matter what the truth value is, these two formulas come out to the same truth value, and that's what makes them equivalent.

Let's look at an important example known as De Morgan's law. So De Morgan's law says that if I look at this formula, the NOT of P or Q, the negation of P or Q, I claim that that's equivalent to not P and not Q, and let's check that with a truth table.

So here's going to be the truth table for the first formula, NOT P or Q, so let's write out the four possible values for P and Q. These are the four possible environments, one per row, and there is the formula whose value I'm trying to compute.

So we do it in the usual way. I'm not going to repeat the truth values of P and Q because they're given here, but I can fill in the values of the OR because I know the truth table for OR and I discovered that the first three rows or subformula is true and the last place is false when both of them are false. And that means that I know the value of the whole formula, which is the negation. The NOT of all those values just flips the trues and falses, and that is the final truth values for this NOT P or Q in all possible environments.

Let's do the same thing for NOT P and NOT Q. Well, this time, I'll fill in the values of NOT P and NOT Q because they're not just repeated there. So the NOT P is the flip of the P column and the NOT Q is the flip of the Q column, and now I can fill in the values at the AND. And of course, the AND is going to be true only when they're both true, and otherwise it's going to be false, and look what I've got.

The possible truth values of the first formula, NOT P or Q, in all possible environments is exactly the same as the possible truth values of NOT P and NOT Q in those corresponding environments. The columns are the same, and that means these two formulas are equivalent, which is the proof by truth table. We've just examined all possible environments and verified that, in fact, they get the same truth value.

This brings us to a useful other connective we haven't talked about yet called if and only if. So the value of the if and only if connective P if and only if Q is true, if and only if P and Q have the same truth value.

Now, this if and only if is an English word that you're supposed to understand what it means, and that if and only if is an operator on truth values that we're defining. Since the English now can be confusing between that if and only if and this if and only if, let's disambiguate by having the truth table for if and only if. Here it is.

What it says is that when both of them are true-- P if and only if Q is true-- and both of them are false-- P if and only if Q is true-- and otherwise, it's false. You can check that P if and only if is true exactly when the complement of P XOR Q is true. So now we come to two crucial properties of formulas called satisfiability and validity, and let's examine what those are.

So a formula is satisfiable if and only if it's true in some environment. That is, it's satisfiable if there is some way to set the values of the variables P and Q to be truth values in such a way that the formula comes out to be true. And a related idea is that a formula is valid-- it's also called a tautology-- if and only if it's true in all environments. No matter what you set the variables to, it's going to come out to be true.

Let's look at some examples to solidify those two concepts. So the formula P all by itself is satisfiable because it can be true if P is true, but it's not always true because P might be false. Symmetrically, NOT P is also satisfiable because it could be true if P is false, but it's not always true. It's not valid because P might be true, in which case not P would be false.

A formula that's not satisfiable, a formula that means that there is no truth value that makes it true, which is the same as saying that it's always false, is the formula P and NOT P. It's probably the simplest not satisfiable formula or unsatisfiable formula.

So this is clearly false because either P or NOT P has got to be false and the [? and ?] will then will definitely come out to be false. There is no value for P that makes this formula true.

It's unsatisfiable.

A valid formula, actually by De Morgan's law applied to the P and NOT P, is P or NOT P is going to be valid because no matter what truth value P has, it comes out to be true. That is, one of P and NOT P is true, and therefore the OR is going to get at least one true, exactly one true really, and going to come out to be true, so this is valid.

Now we can connect up the pieces that we've just set up of relating validity and equivalence. Two formulas G and H are equivalent is the same as saying that G if and only if H is valid. So G if and only if H comes out to be true when G and H have the same truth value, and G and H are equivalent says that they do have the same truth value no matter what the environment is.

So that's the same as saying that if G and H are equivalent no matter what the environment is, G if and only if H comes out to be true. So if G and H are equivalent, G if and only if H is valid, and the converse argument works the same way.

A very important problem that comes up in multiple ways, and we're going to examine some of them later, is the problem of whether or not a formula is valid-- proving that it's valid-- and checking whether or not a formula is satisfiable. Now, there's a simple way to do that. The truth table tells it to you. If you want to know whether the formula is satisfiable, you just look at its truth table. Try every possible environment and see whether one of them yields the value of true.

The problem with that approach, which theoretically is sound, but pragmatically the truth table size doubles with each additional variable. So with two variables, you got four rows. With three variables, you got eight rows. With four variables, you got 16 rows, and this very rapidly gets out of hand once the number of variables gets to be moderate sized. This is exponential growth. It's doubling each time you add a variable.

So really when you start having hundreds of variables, truth tables are out of the question. You just can't write them down. They're too big, and in fact, in modern digital circuits when you think back about how we designed the [? adder, ?] if you look at what's going on in all those different wires corresponding to different variables, there are millions of those in a typical modern digital circuit. So truth table approach is just not going to be workable. It's hopeless.

Now, one of the central problems in theoretical computer science is the question of whether or not there is a way to test for SAT that's more efficient than this impossible way of trying to

come up with a truth table that's too big to fit in the universe when there are hundreds and thousands of variables. So we're interested in the question of is there some other way to check a formula for satisfiability than exhaustively trying to generate its whole truth table to see if some row yields true?

This is the abstract version of the P equals NP problem. So we've talked about this before. P equals NP? is considered to be the most important open problem in theoretical computer science, in the theory of computation, and it's simply saying, is there some fast or efficient way to tell whether or not a formula is satisfiable that's more efficient than the truth table approach?

Efficiency has a technical definition, which is that the number of steps grows much less than exponentially with the number of variables. Let's say it grows polynomial, like a quadratic or a cubic, but you're trying to beat exponential growth, which is what ruins things quickly. And this is an open problem. No one knows if there is some fast way to check for satisfiability or SAT. That's the SAT problem.

By the way, very closely related to SAT is validity because if I wanted to know whether a formula G is valid-- well, valid means that it's always true. That means its complement, not G, is always false, which is the same as saying its complement is not satisfiable. So to check that G is valid, all I need to do is check that not G is satisfiable and vice versa, not G is not satisfiable, if and only if G is valid.

So the point is that SAT and valid stand and fall together. If you had a fast way to do one, you would very quickly get a fast way to do the other one. So checking for one is just as difficult as checking for the other, and we're going to examine in subsequent lectures why it is that this problem is so important.