[1] A quantifier is just an operator that applies to a function. [2] The upside down A is called the universal quantifier, pronounced “for all”. It applies to functions that have a binary result; in other words, it applies to predicates. And the result of applying a universal quantifier to a predicate is binary. The result can be thought of as applying the predicate to all its domain elements, and then conjoining all those results together. [3] Here's an example. For all r in rat, r is less than 0 or equal to 0 or greater than 0. In this example, the result is true. [4] The backwards E is called the existential quantifier, pronounced “there exists”. In my opinion, that's a very unfortunate pronunciation, because it has nothing to do with the existence of anything. Like for all, it's an operator that applies to predicates, with a binary result. The result can be thought of as applying the predicate to all its domain elements, and then disjoining all those results together. [5] Here's an example. There exists n in nat such that n equals 0. In this example, the result is again true. [6] The Greek capital sigma is the summation quantifier. It applies to functions that have a numeric result, and its result is numeric. The result can be thought of as applying the function to all its domain elements, and then adding all those results together. [7] Here's an example. The sum, as n varies over nat plus 1, of 1 over 2 to the n. So that's a half plus a quarter plus an eighth, and so on, and the result is 1. [8] The Greek capital P is the product quantifier. It applies to functions that have a numeric result, and its result is numeric. The result can be thought of as applying the function to all its domain elements, and then multiplying all those results together. [9] Here's an example. The product, as n varies over nat plus 1, of – that thing. The result here, by the way, is pi over 2.

[10] There are a couple of slight abbreviations we will use. The first is [11] to stop writing the scope brackets when a quantifier is applied to a function, and to write a raised dot instead of an arrow. I really don't like this abbreviation at all. The scope brackets are useful for showing the scope. But we're moving to tradition here, and the tradition is to write the quantifier beside the variable. And the scope is everything to the right, up to a big equals or big implies sign, or the end of the expression, or an enclosing parenthesis. The other abbreviation [12] is to group the variables when there is a repeated quantification with the same domain. For all x and y in rat, something [13] really means for all x in rat, for all y in rat, something. [14] For another example, the sum as n and m vary over 0 to 10 of n times m [15] really means the sum as n varies over 0 to 10 of the sum as m varies over 0 to 10 of n times m.

[16] Here are the axioms defining these quantifiers. I've put them all on one slide so you can see the pattern. The first axiom for each quantifier shows its effect on the null domain. The middle axiom for each quantifier has a one element domain, and the last axiom for each quantifier has a union domain. [17] For the null domain, the result is always the identity element for the operator that the quantifier is based on. True is the identity for conjunction, false is the identity for disjunction, 0 is the identity for addition, and 1 is the identity for multiplication. Most people don't have any trouble with [18] this one. The sum of no numbers is 0. If you're adding things up, you initialize the variable to 0. [19] If you're multiplying things together, you have to initialize the variable to 1. So the product of no numbers is 1. [20] You probably agree that there isn't an element in the empty domain with property b. And you've probably written a search loop that began by initializing found to false. The disjunction of no binary values is false. But you might have trouble with [21] this one. All elements in the empty domain have property b, whatever b is. Well, if it helps, you can say that there aren't any elements in the empty domain that don't have property b. So they all have it. All zero of them. If that helps. Anyway, the conjunction of no binary values is true. [22] For the one element domain, the result is always the result of applying the function to that one element. The sum of one number is that number. And so on. [23] For a union domain, that's where the operator that the quantifier is based on comes in. For all v in the union of A and B means for all v in A and for all v in B. [24] There exists v in the union

means there exists v in A or there exists v in B. [25] Sum and product are slightly different from for all and there exists. That's because conjunction and disjunction are idempotent, but sum and product aren't. To add up over the union of A and B, you have to add over A, and add over B, but then you might have added some things twice, so you have to subtract things over the intersection. And [26] product is similar.

The quantifiers we've just seen are standard ones, but [27] anyone can define their own new quantifiers. A quantifier is just an operator that applies to a function. So, for example, we can define the [28] MAX quantifier based on the max function that we defined last lecture. And we can apply our new quantifier to a function like [29] this one. And, to be consistent with the other quantifiers, we can abbreviate by [30] leaving out the scope brackets and changing the arrow to a raised dot. This is the maximum, as x varies over the rationals, of 4 x minus x squared. Which, by the way is [31] 4. To define MAX, [32] we need to say what its effect is on an empty domain, on a one element domain, and on a union of domains. [33] The one element domain is easy. It's the result of applying the function to that one element. And the [34] union domain is easy. Just find the MAX over each part of the union, and then take the max of those two results. But what do we do for the empty domain? For the [35] previous quantifiers, we used the identity element. So we need [36] some element such that the max of x and that element is always x, no matter what x is. So it has to be less than or equal to all x. It has to be [37] minus infinity. In a program with a loop for finding max, you would have to initialize a variable to the most negative number on your computer.

There's one last quantifier that we need. [38] The solution quantifier. It applies to a predicate and gives its solutions. If the predicate has an empty domain, [39] there are no solutions in that domain. If the predicate has a one element domain, that one element may or may not be a solution. [40] If it is, then the result is that element. If it isn't, then there are no solutions of the predicate in that domain. And when the domain is a union [41] the solutions are the union of the solutions for the parts of the union. [42] Here's an example. I pronounce it: those i in int such that i squared equals 4. And the result is [43] minus 2 and 2. It's the formal way of saying what variable we're solving for, and what domain we're looking in. We're solving an equation here, but an equation is just a special case of binary expression. This [44] next example solves an inequality. Those n in nat that are less than 3. And the result is [45] 0 1 and 2. These are bunch equations, and if we [46] apply set formation, we get set equations. And once again, just for the sake of allowing traditional mathematical notations, [47] we can leave out the solutions quantifier in this one context. I'm sure you've used this quantifier before, at least in this context, but maybe you didn't know you were using it. There are lots of laws about quantifiers. So have a look in the back of the textbook and see what they are.

I want to end this lecture with a very important message. [48] An expression talks about its nonlocal variables. This expression can be read as: there exists an n in nat such that x equals 2 times n. And that's true if and only if x is an even natural. So that's what it's saying. It's saying something about x. The n is there just to help say that. It can't be talking about n, because we could rename n to some other variable and get an equivalent expression. So an expression talks about its nonlocal variables.