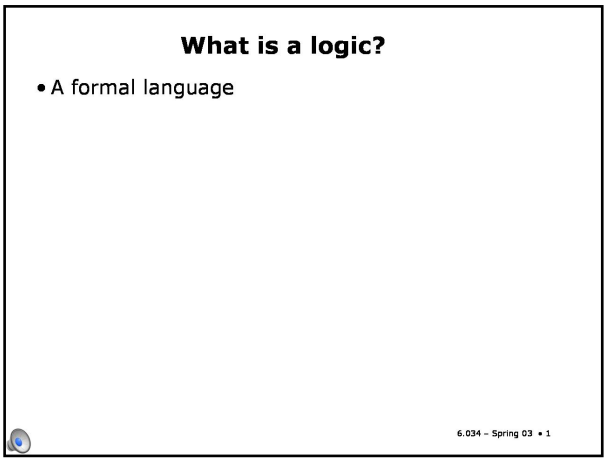
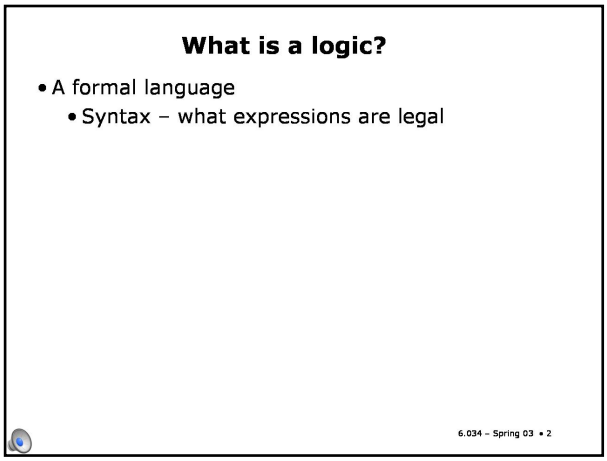
6.034 Notes: Section 7.1

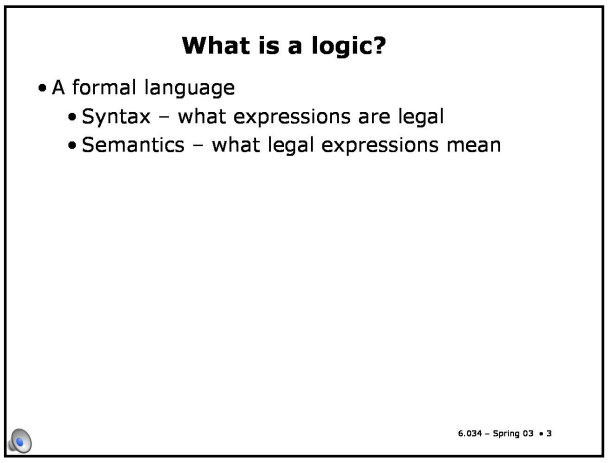
Slide 7.1.1
What is a logic? A logic is a formal language. And what does that mean? It has a syntax and a semantics, and a way of manipulating expressions in the language. We'll talk about each of these in turn.

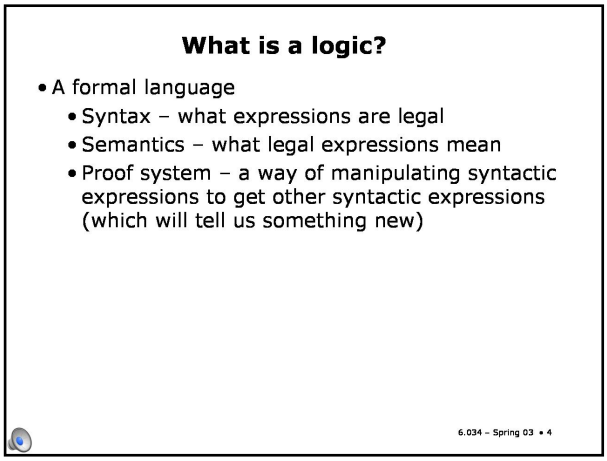




Slide 7.1.2
The syntax is a description of what you're allowed to write down; what the expressions are that are legal in a language. We'll define the syntax of a propositional logic in complete detail later in this section.

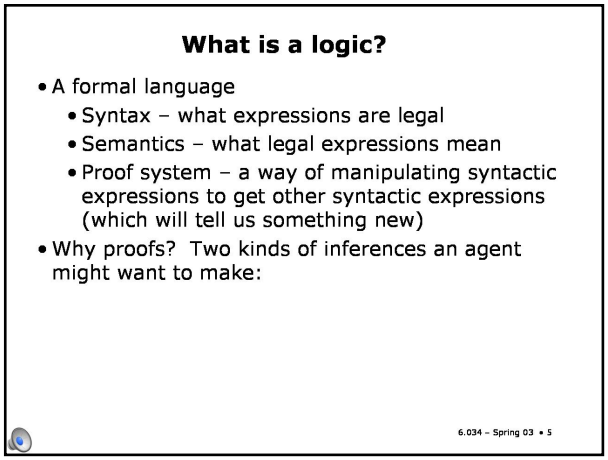
Slide 7.1.3
The semantics is a story about what the syntactic expressions mean. Syntax is form and semantics is content.

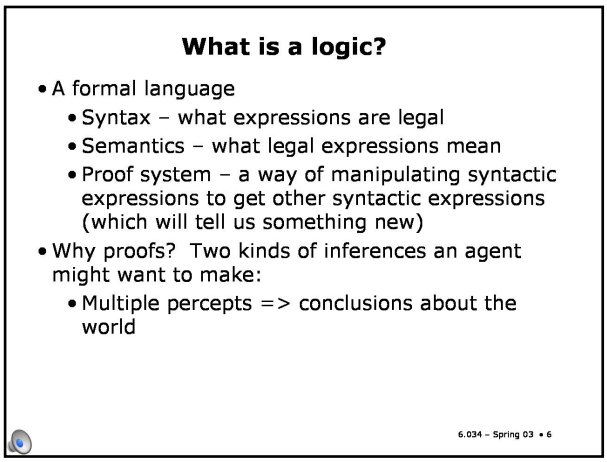




Slide 7.1.4
A logic usually comes with a proof system, which is a way of manipulating syntactic expressions to get other syntactic expressions. And, why are we interested in manipulating syntactic expressions? The idea is that if we use a proof system with the right kinds of properties, then the new syntactic expressions we create will have semantics or meanings that tell us something "new" about the world.

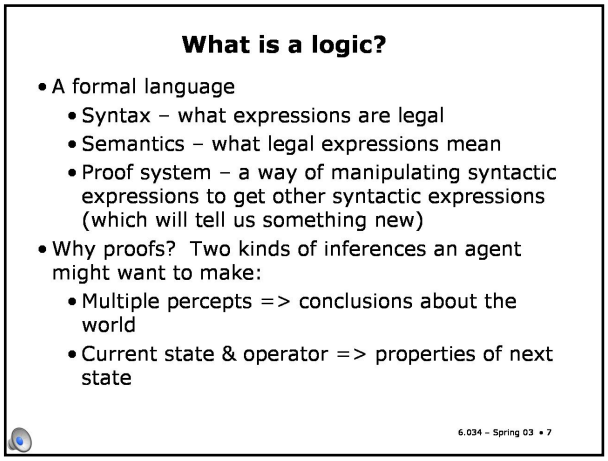
Slide 7.1.5
So, why would we want to do proofs? There are lots of situations.

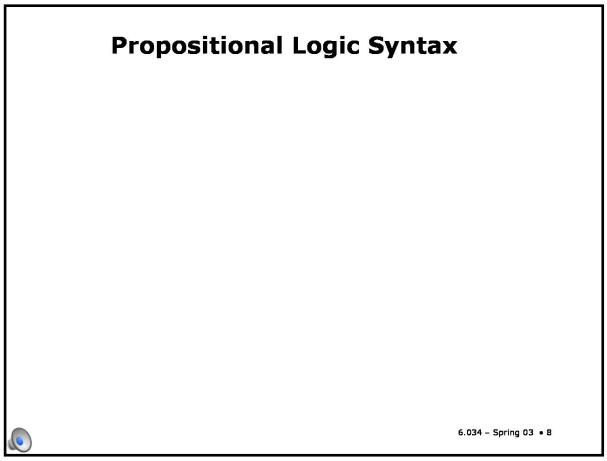




Slide 7.1.6
In the context of an agent trying to reason about its world, think about a situation where we have a bunch of percepts. Let's say we saw somebody come in with a dripping umbrella, we saw muddy tracks in the hallway, we see that there's not much light coming in the windows, we hear pitter-pitter-patter. We have all these percepts, and we'd like to draw some conclusion from them, meaning that we'd like to figure out something about what's going on in the world. We'd like to take all these percepts together and draw some conclusion about the world. We could use logic to do that.

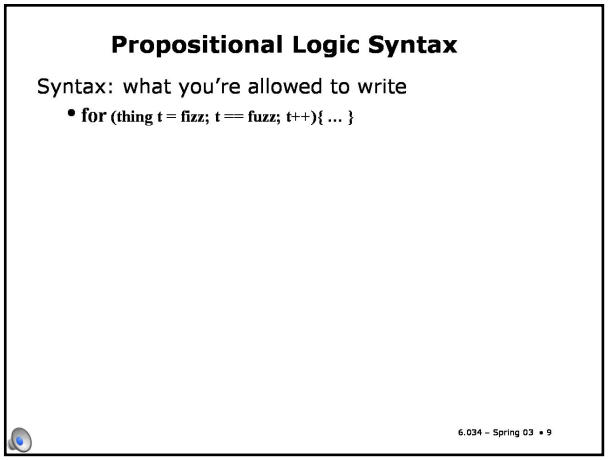
Slide 7.1.7
Another use of logic is when you know something about the current state of the world and you know something about the effects of an action that you're considering doing. You wonder what will happen if you take that action. You have a formal description of what that action does in the world. You might want to take those things together and infer something about the next state of the world. So these are two kinds of inferences that an agent might want to do. We could come up with a lot of other ones, but those are two good examples to keep in mind.

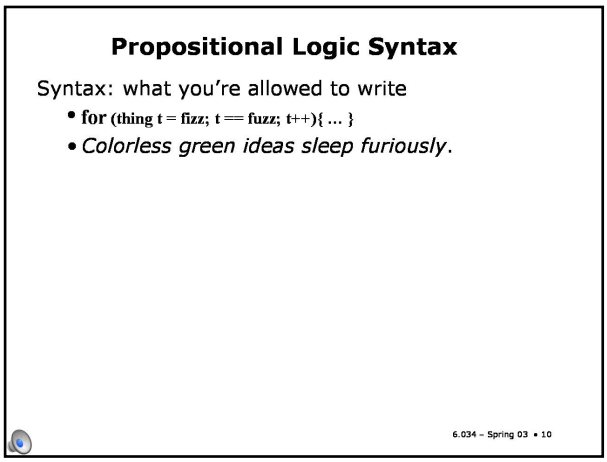




Slide 7.1.8
We'll look at two kinds of logic: propositional logic, which is relatively simple, and first-order logic, which is more complicated. We're just going to dive right into propositional logic, learn something about how that works, and then try to generalize later on. We'll start by talking about the syntax of propositional logic. Syntax is what you're allowed to write on your paper.

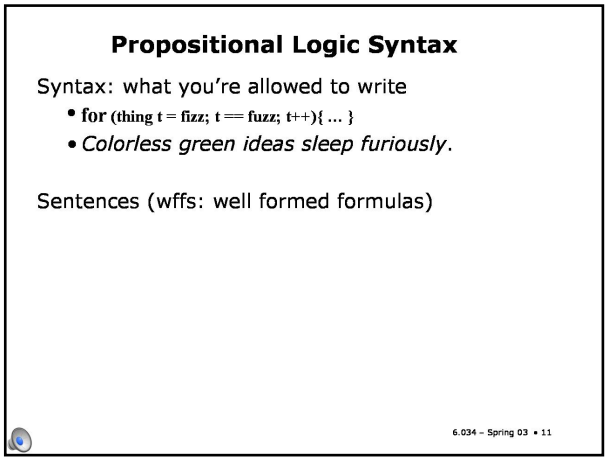
Slide 7.1.9
You're all used to rules of syntax from programming languages, right? In Java you can write a for loop. There are rules of syntax given by a formal grammar. They tell you there has to be a semicolon after fizz; that the parentheses have to match, and so on. You can't make random changes to the characters in your program and expect the compiler to be able to interpret it. So, the syntax is what symbols you're allowed to write down in what order. Not what they mean, not what computation they symbolize, but just what symbols you can write down.

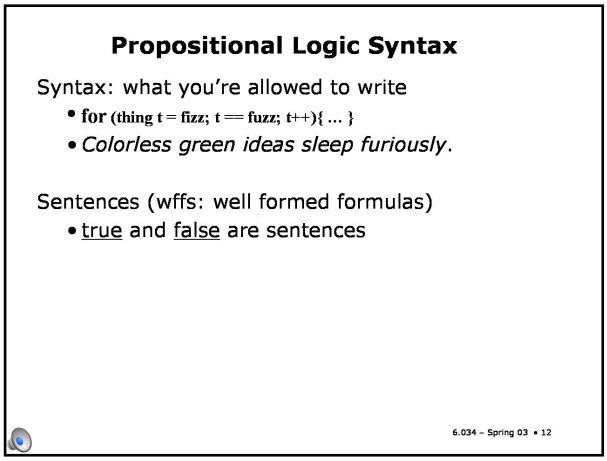




Slide 7.1.10
Another famous illustration of syntax is this one, due to the linguist Noam Chomsky: "Colorless green ideas sleep furiously". The idea is that it doesn't mean anything really, but it's syntactically well-formed. It's got the nouns, the verbs, and the adjectives in the right places. If you scrambled the words up, you wouldn't get a sentence, right? You'd just get a string of words that didn't obey the rules of syntax. So, "furiously ideas green sleep colorless" is not syntactically okay.

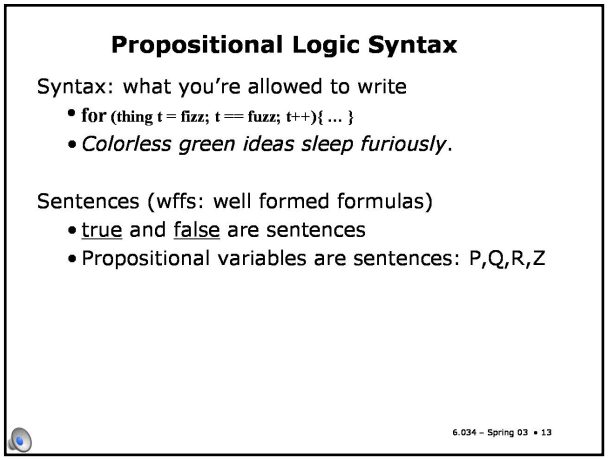
Slide 7.1.11
Let's define the syntax of propositional logic. We'll call the legal things to write down "sentences". So if something is a sentence, it is a syntactically okay thing in our language. Sometimes sentences are called "WFFs" (which stands for "well-formed formulas") in other books.

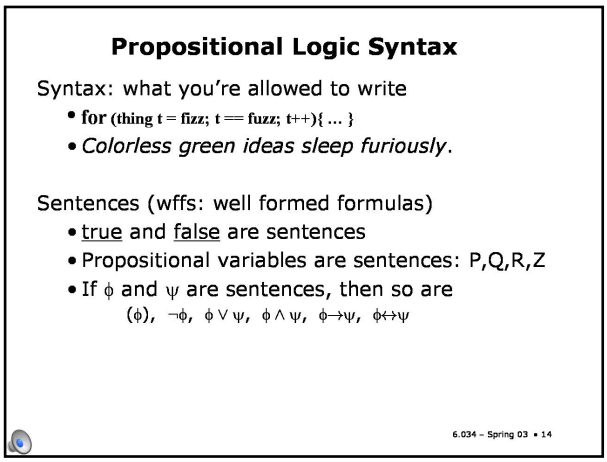




Slide 7.1.12
We're going to define the set of legal sentences recursively. So here are two base cases: The words, "true" and "false", are sentences.

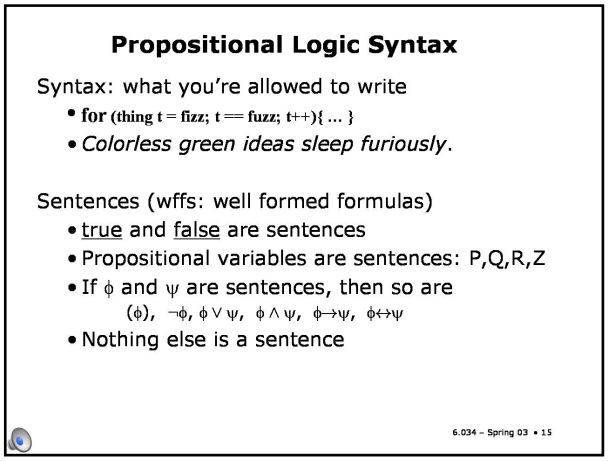
Slide 7.1.13
Propositional variables are sentences. I'll give you some examples. P, Q, R, Z. We're not, for right now, defining a language that a computer is going to read. And so we don't have to be absolutely rigorous about what characters are allowed in the name of a variable. But there are going to be things called variables, and we'll just use uppercase letters for them. Those are sentences. It's OK to say "P" -- that's a sentence.

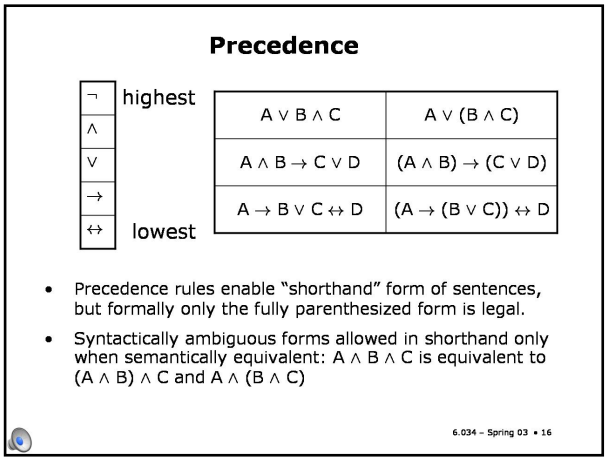




Slide 7.1.14
Now, here's the recursive part. If Phi and Psi are sentences, then so are -- Wait! What, exactly, are Phi and Psi? They're called metavariables, and they range over expressions. This rule says that if Phi and Psi are things that you already know are sentences because of one of these rules, then you can make more sentences out of them. Phi with parentheses around it is a sentence. Not Phi is a sentence (that little bent thing is our "not" symbol (but we're not really supposed to know that yet, because we're just doing syntax right now)). Phi "vee" Psi is a sentence. Phi "wedge" Psi is a sentence. Phi "arrow" Psi is a sentence. Phi "two-headed arrow" Psi is a sentence.

Slide 7.1.15
And there's one more part of the definition, which says nothing else is a sentence. OK. That's the syntax of the language.

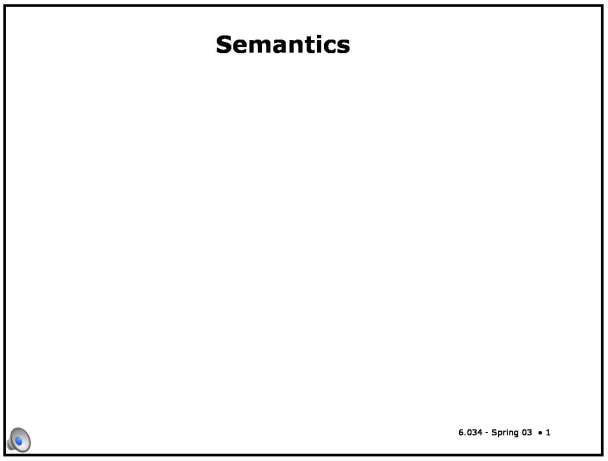


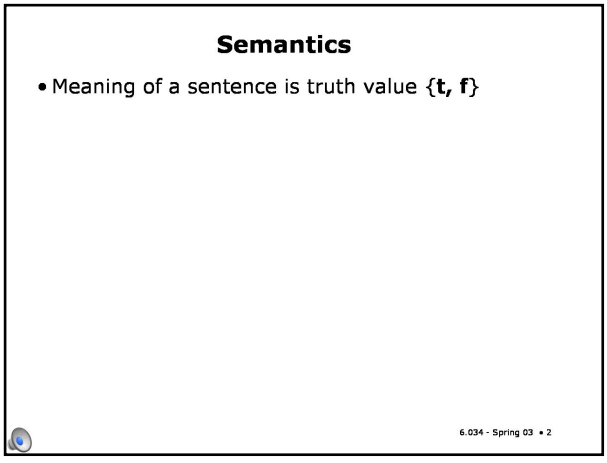


Slide 7.1.16
There's actually one more issue we have to sort out. Precedence of the operations. If we were being really careful, we'd require you to put parentheses around each new sentence that you made out of component sentences using negation, vee, wedge, or arrow. But it starts getting kind of ugly if we do that. So, we allow you to leave out some of the parentheses, but then we need rules to figure out where the implicit parentheses really are. Those are precedence rules. Just as in arithmetic, where we learned that multiplication binds tighter than addition, we have similar rules in logic. So, to add the parentheses to a sentence, you start with the highest precedence operator, which is negation. For every negation, you'd add an open paren in front of the negation sign and a close parenthesis after the next whole expression. This is exactly how minus behaves in arithmetic. The next highest operator is wedge, which behaves like multiplication in arithmetic. Next is vee, which behaves like addition in arithmetic. Logic has two more operators, with weaker precedence. Next comes single arrow, and last is double arrow. Also, wedge and vee are associative.

6.034 Notes: Section 7.2

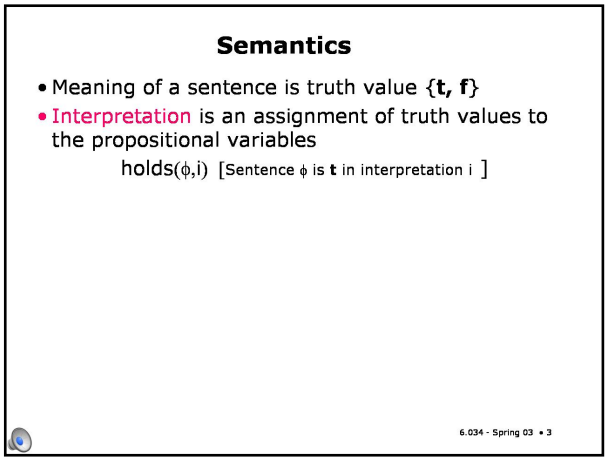
Slide 7.2.1
Let's talk about semantics. The semantics of a sentence is its meaning. What does it say about the world? We could just write symbols on the board and play with them all day long, and it could be fun; it could be like doing puzzles. But ultimately the reason that we want to be doing something with these kinds of logical sentences is because they somehow say something about the world. And it's really important to be clear about the connections between the things that we write on the board and what we think of them as meaning in the world, what they stand for. And it's going to be something different every day. I remember once when I was a little kid, I was on the school bus. And somebody's big sister or brother had started taking algebra and this kid told me, "You know what? My big sister's taking algebra and A equals 3!" The reason that sounds so silly is that A is a variable. Our variables are going to be the same. They'll have different interpretations in different situations. So, in our study of logic, we're not going to assign particular values or meanings to the variables; rather, we're going to study the general properties of symbols and their potential meanings.

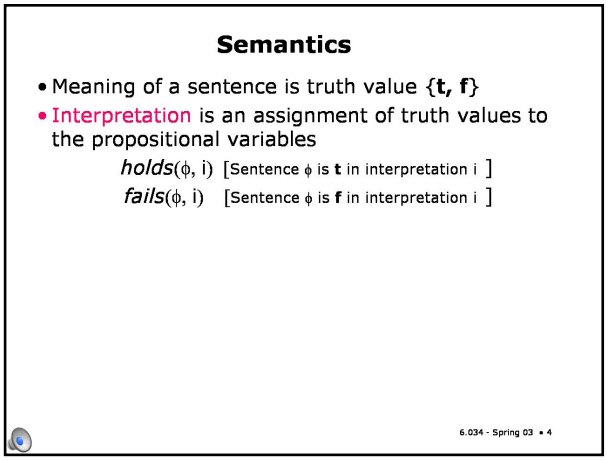




Slide 7.2.2
Ultimately, the meaning of every sentence, in a situation, will be a truth value, **t** or **f**. Just as, in high-school algebra, the meaning of every expression is a numeric value. Note that there's already a really important difference between underlined true and false, which are syntactic entities that we can write on the board, and the truth values **t** and **f** which stand for the abstract philosophical ideals of truth and falsity.

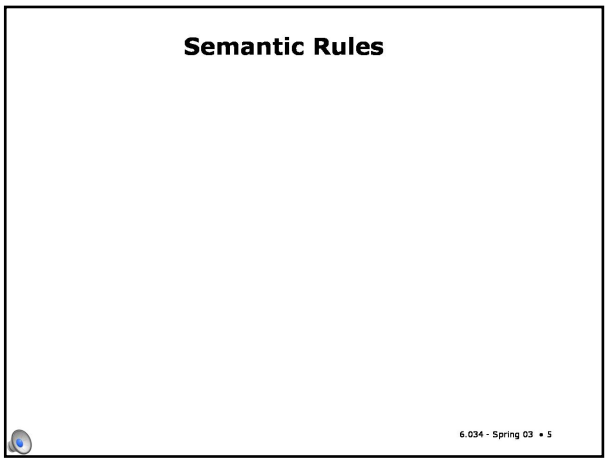
Slide 7.2.3
How can we decide whether A "wedge" B "wedge" C is true or not? Well, it has to do with what A and B and C stand for in the world. What A and B and C stand for in the world will be given by an object called an "interpretation". An interpretation is an assignment of truth values to the propositional variables. You can think of it as a possible way the world could be. So if our set of variables is P, Q, R, and V, then P true, Q false, R true, V true, that would be an interpretation. So then, given an interpretation, we can ask the question, is this sentence true in that interpretation? We will write "holds Phi comma i" to mean "sentence Phi is true in interpretation i". The "holds" symbol is not part of our language. It's part of the way logicians write things on the board when they're talking about what they're doing. This is a really important distinction. If you can think of our sentences like expressions in a programming language, then you can think of these expressions with "holds" as being about whether programs work in a certain way or not. In order to even think about whether Phi is true in interpretation I, Phi has to be a sentence. If it's not a well-formed sentence, then it doesn't even make sense to ask whether it's true or false.

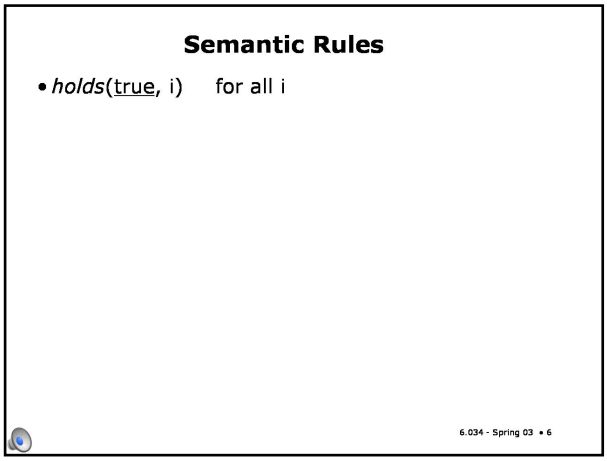




Slide 7.2.4
Similarly, we'll use "fails" to say that a sentence is not true in an interpretation. And since the meaning of every sentence is a truth value and there are only two truth values, then if a sentence Phi is not true (does not have the truth value **t**) in an interpretation, then it has truth value **f** in that interpretation and we'll say it's false in that interpretation.

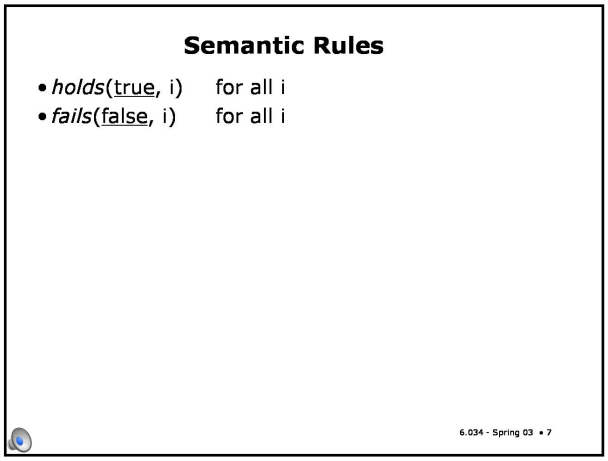
Slide 7.2.5
So now we can write down the rules of the semantics. We can write down rules that specify when sentence Phi is true in interpretation i. We are going to specify the semantics of sentences recursively, based on their syntax. The definition of a semantics should look familiar to most of you, since it's very much like the specification of an evaluator for a functional programming language, such as Scheme.

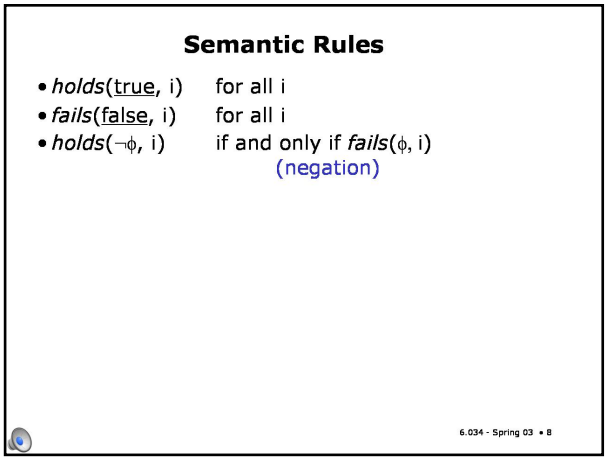




Slide 7.2.6
First, the sentence consisting of the symbol "true" is true in all interpretations.

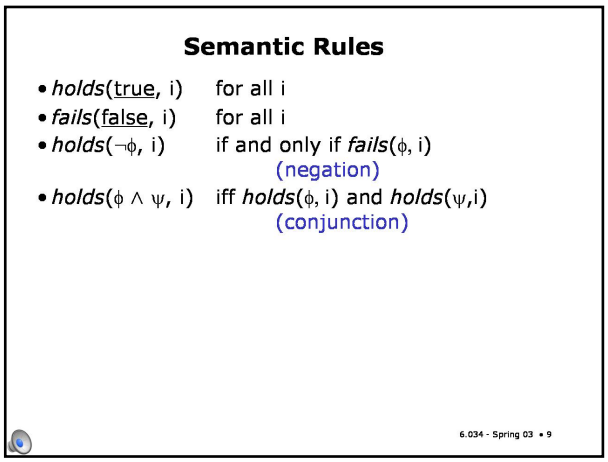
Slide 7.2.7
The sentence consisting of a symbol "false" has truth value **f** in all interpretations.

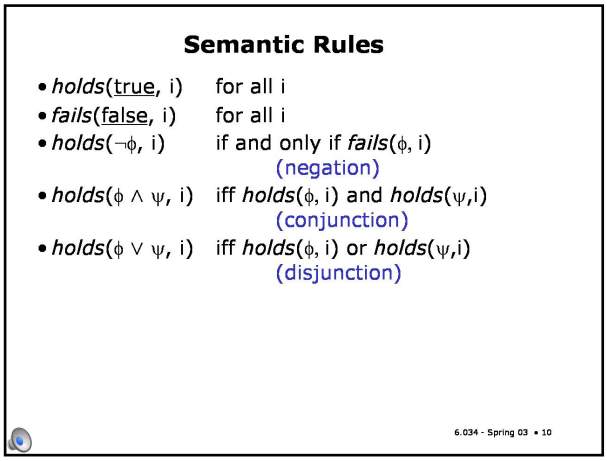




Slide 7.2.8
Now we can do the connectives. We'll leave out the parentheses. The truth value of a sentence with top-level parentheses is the same as the truth value of the sentence with the parentheses removed. Now, let's think about the "not" sign. When is "not" Phi true in an interpretation i? Whenever Phi is false in that interpretation.

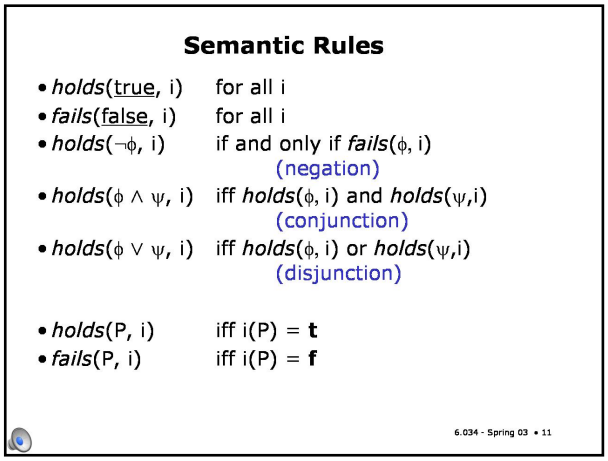
Slide 7.2.9
When is Phi "wedge" Psi true in an interpretation i? Whenever both Phi and Psi are true in i. This is called "conjunction". And we'll start calling that symbol "and" instead of "wedge", now that we know what it means.

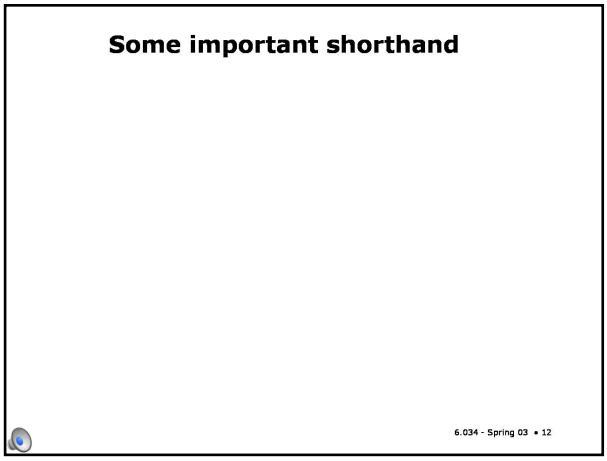




Slide 7.2.10
When is Phi "vee" Psi true in an interpretation i? Whenever either Phi or Psi is true in i. This is called "disjuction", and we'll call the "vee" symbol "or". It is not an exclusive or; so that if both Phi and Psi are true in i, then Phi "vee" Psi is also true in i.

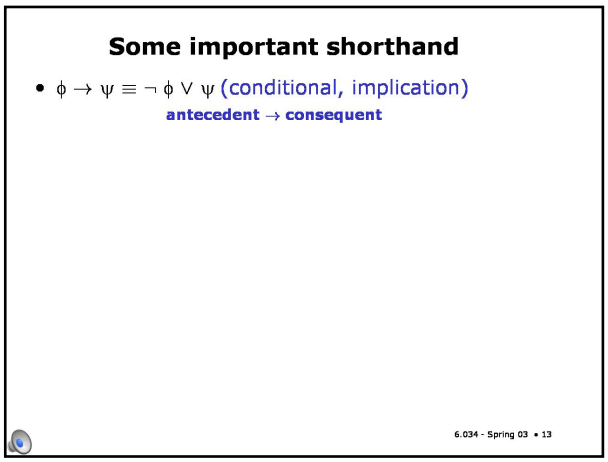
Slide 7.2.11
Now we have one more clause in our definition. I'm going to do it by example. Imagine that we have a sentence P. P is one of our propositional variables. How do we know whether it is true in interpretation i? Well, since i is a mapping from variables to truth values, I can simply look P up in i and return whatever truth value was assigned to P by i.

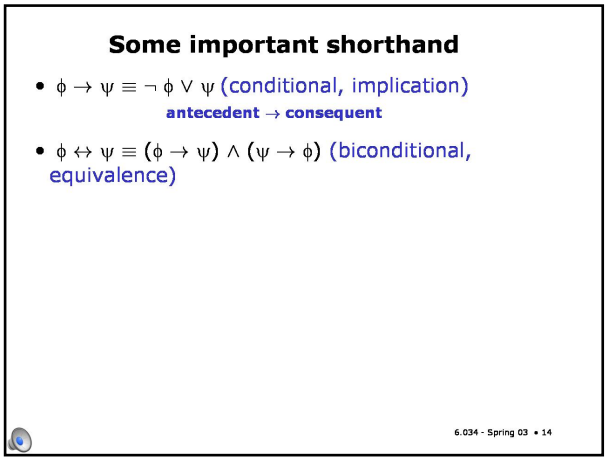




Slide 7.2.12
It seems like we left out the arrows in the semantic definitions of the previous slide. But the arrows are not strictly necessary; that is, it's going to turn out that you can say anything you want to without them, but they're a convenient shorthand. (In fact, you can also do without either "or" or "and", but we'll see that later).

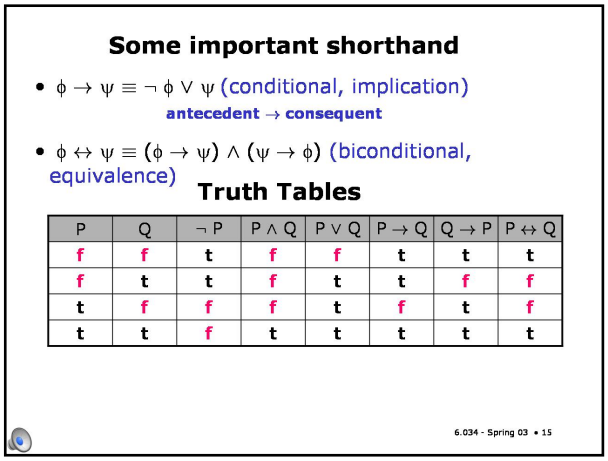
Slide 7.2.13
So, we can define Phi "arrow" Psi as being equivalent to not Phi or Psi. That is, no matter what Phi and Psi are, and in every interpretation, (Phi "arrow" Psi) will have the same truth value as (not Phi or Psi). We will now call this arrow relationship "implication". We'll say that Phi implies Psi. We may also call this a conditional expression: Psi is true if Phi is true. In such a statement, Phi is referred to as the antecedent and Psi as the consequent.



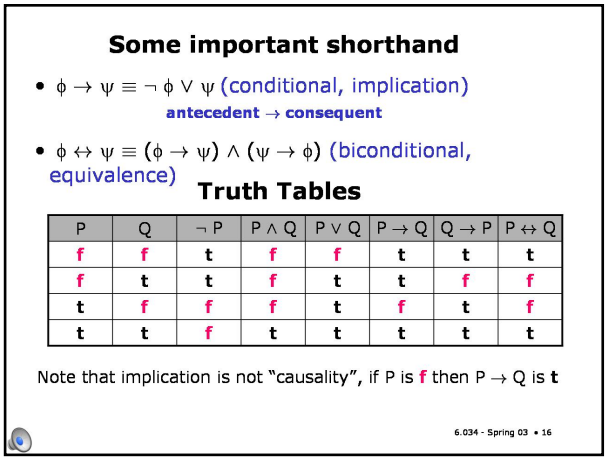


Slide 7.2.14
Finally, the double arrow just means that we have single arrows going both ways. This is sometimes called a "bi-conditional" or "equivalence" statement. It means that in every interpretation, Phi and Psi have the same truth value.

Slide 7.2.15
Just so you can see how all of these operators work, here are the truth tables. Consider a world with two propositional variables, P and Q. There are four possible interpretations in such a world (one for every combination of assignments to the variables; in general, in a world with n variables, there will be 2^n possible interpretations). Each row of the truth table corresponds to a possible interpretation, and we've filled in the values it assigns to P and Q in the first two columns. Once we have chosen an interpretation (a row in the table), then the semantic rules tell us exactly what the truth value of every single legal sentence must be. Here we show the truth values for six different sentences made up from P and Q.



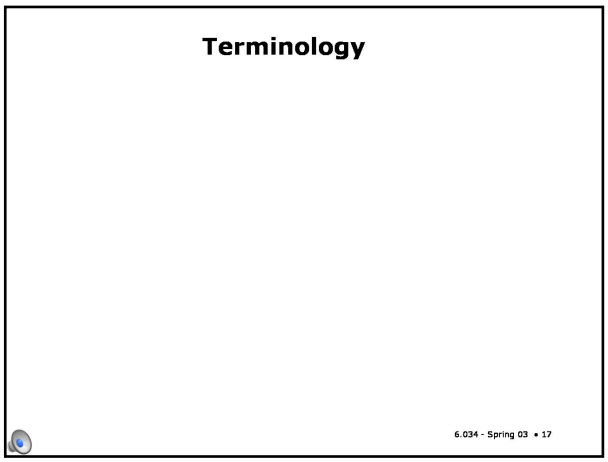
P	Q	¬ P	P ∧ Q	P ∨ Q	P → Q	Q → P	P ↔ Q
f	f	t	f	f	t	t	t
f	t	t	f	t	t	f	f
t	f	f	f	t	f	t	f
t	t	f	t	t	t	t	t

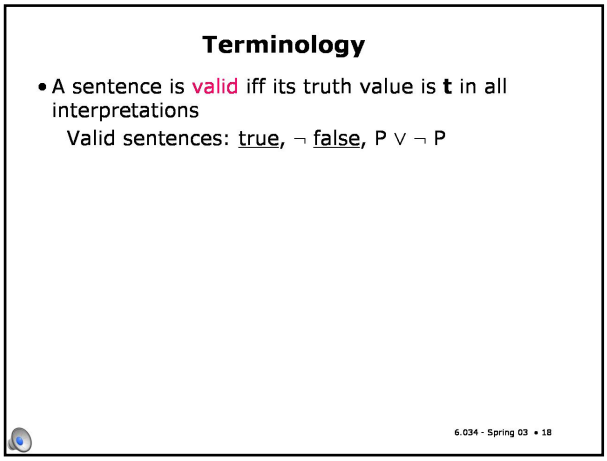


P	Q	¬ P	P ∧ Q	P ∨ Q	P → Q	Q → P	P ↔ Q
f	f	t	f	f	t	t	t
f	t	t	f	t	t	f	f
t	f	f	f	t	f	t	f
t	t	f	t	t	t	t	t

Slide 7.2.16
Most of them are fairly obvious, but it's worth studying the truth table for implication fairly closely. In particular, note that (P implies Q) is true whenever P is false. You can see that this is reasonable by thinking about an English sentence like "If pigs can fly then ...". Once you start with a false condition, you can finish with anything, and the sentence will be true. Implication doesn't mean "causes". It doesn't mean "is related" in any kind of real-world way; it is just a bare, formal definition of not P or Q.

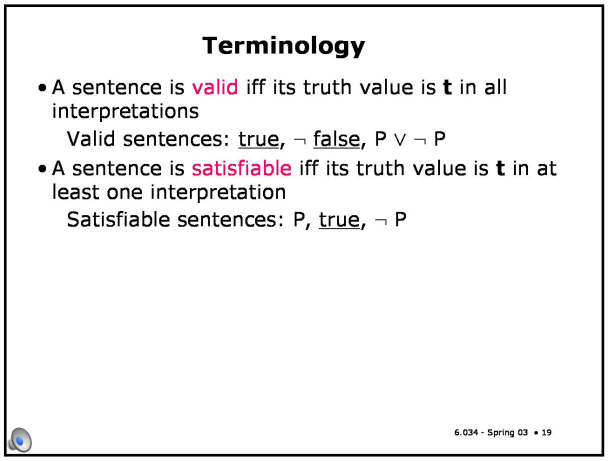
Slide 7.2.17
Now we'll define some terminology on this slide and the next, then do a lot of examples.

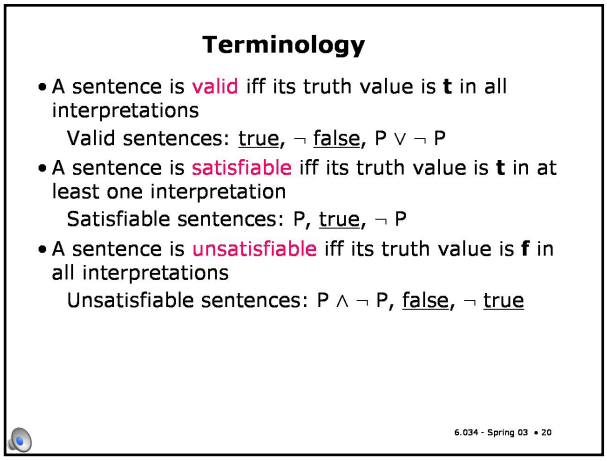




Slide 7.2.18
A sentence is **valid** if and only if it is true in all interpretations. We have already seen one example of a valid sentence. What was it? True. Another one is "not false". A more interesting one is "P or not P". No matter what truth value is assigned to P by the interpretation, "P or not P" is true.

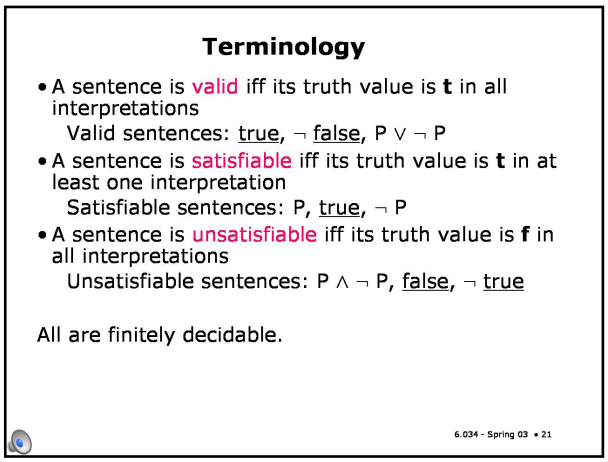
Slide 7.2.19
A sentence is satisfiable if and only if it's true in at least one interpretation. The sentence P is satisfiable. The sentence True is satisfiable. Not P is satisfiable.

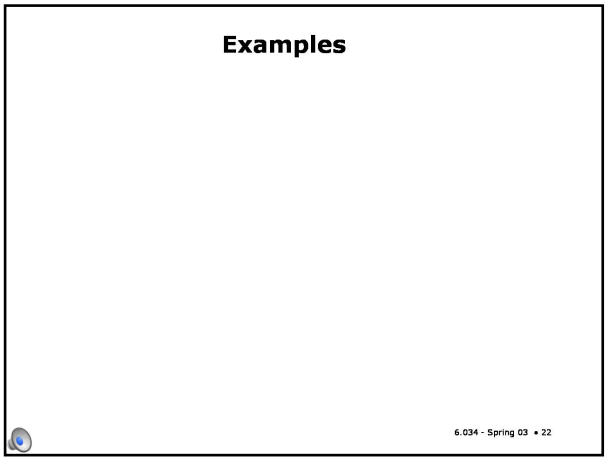




Slide 7.2.20
A sentence is unsatisfiable if and only if it's false in every interpretation. Some unsatisfiable sentences are: false, not true, P and not P.

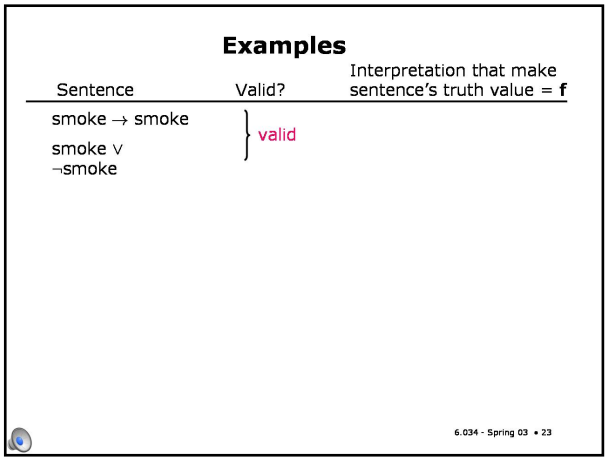
Slide 7.2.21
We can use the method of truth tables to check these things. If I wanted to know whether a particular sentence was valid, or if I wanted to know if it was satisfiable or unsatisfiable, I could just make a truth table. I'd write down all the interpretations, figure out the value of the sentence in each interpretation, and if they're all true, it's valid. If they're all false, it's unsatisfiable. If it's somewhere in between, it's satisfiable. So there's a reliable way; there's a completely dopey, tedious, mechanical way to figure out if a sentence is has one of these properties. That's not true in all logics. This is a useful, special property of propositional logic. It might take you a lot of time, but it's a finite amount of time and you can decide any of these questions.

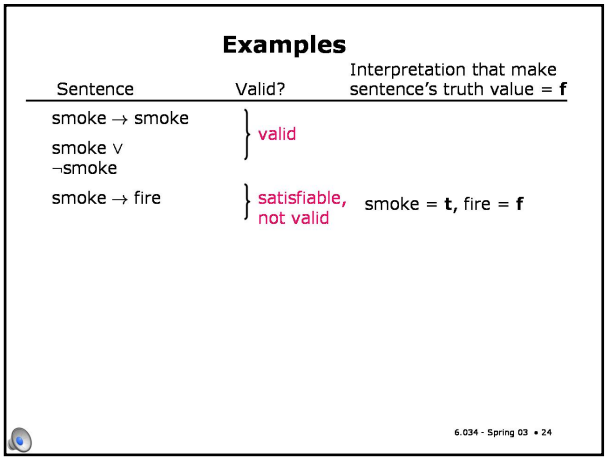




Slide 7.2.22
Let's work through some examples. We can think about whether they're valid or unsatisfiable or satisfiable.

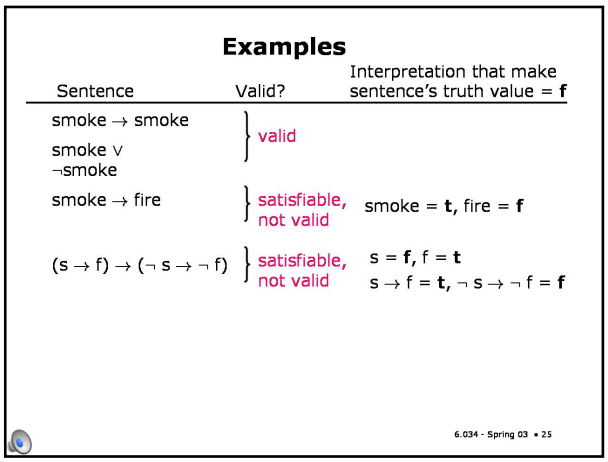
Slide 7.2.23
What about "smoke implies smoke"? Rather than doing a whole truth table it might be easier if we can convert it into smoke or not smoke, right? The definition of A implies B is not A or B. And we said that smoke or not smoke was valid already.

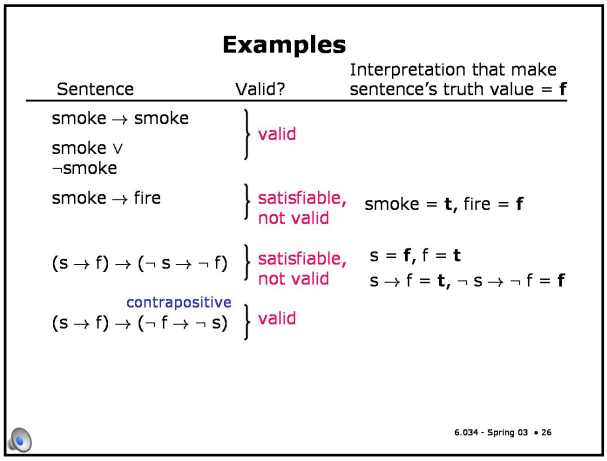




Slide 7.2.24
What about "smoke implies fire"? It's satisfiable, because there's an interpretation of these two symbols that makes it true. There are other interpretations that make it false. I should say, everything that's valid is also satisfiable.

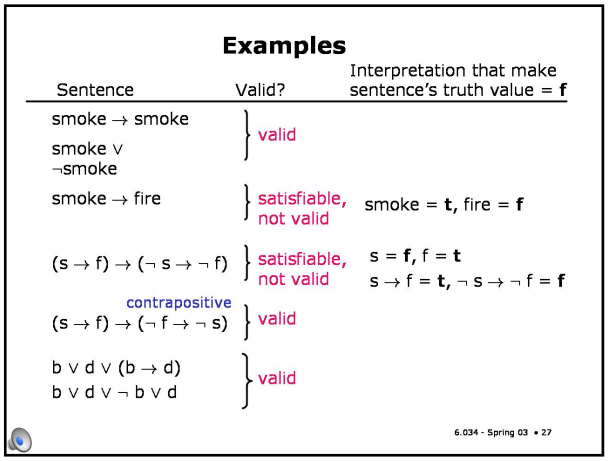
Slide 7.2.25
Here is a form of reasoning that you hear people do a lot, but the question is, is it okay? "Smoke implies fire implies not smoke implies not fire." It's invalid. We could show that by drawing out the truth table (and you should do it as an exercise if the answer is not obvious to you). Another way to show that a sentence is not valid is to give an interpretation that makes the sentence have the truth value **f**. In this case, if we give "smoke" the truth value **f** and and "fire" the truth value **t**, then the whole sentence has truth value **f**.

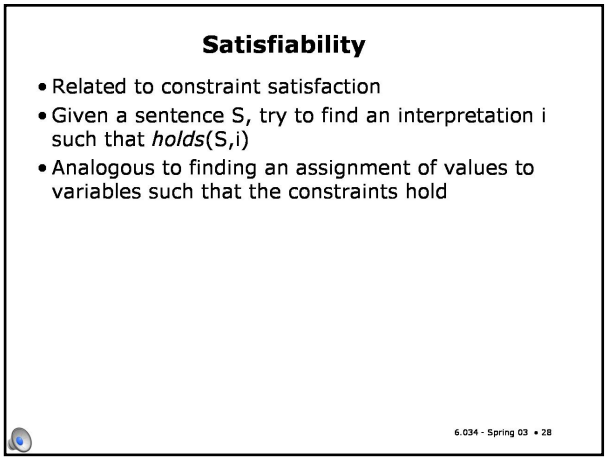




Slide 7.2.26
Reasoning in the other direction is okay, though. So the sentence "smoke implies fire implies not fire implies not smoke" is valid. And for those of you who love terminology, this thing is called the contrapositive. So, if there's no fire, then there's no smoke.

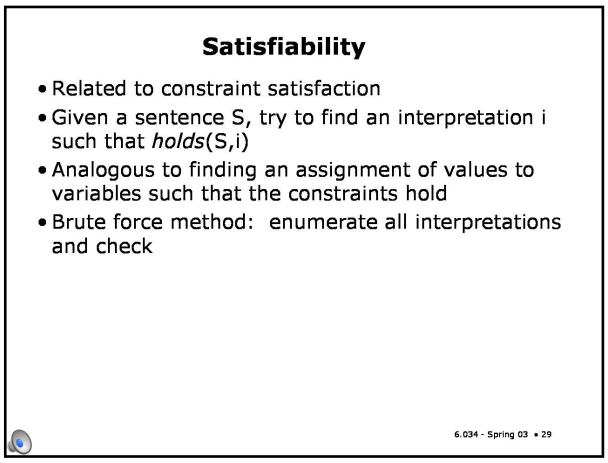
Slide 7.2.27
What about "b or d or (b implies d)"? We can rewrite that (using the definition of implication) into "b or d or not b or d", which is valid, because in every interpretation either b or not b must be true.

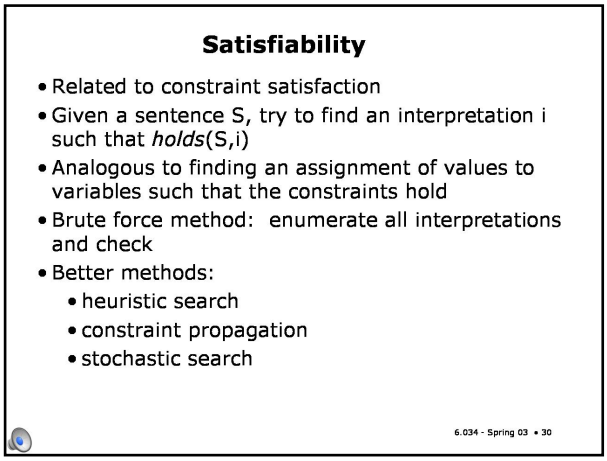




Slide 7.2.28
The problem of deciding whether a sentence is satisfiable is related to constraint satisfaction: you have to find an interpetation i such that the sentence holds in that interpretation. That's analogous to finding an assignment of values to variables so that the constraints are satisfied.

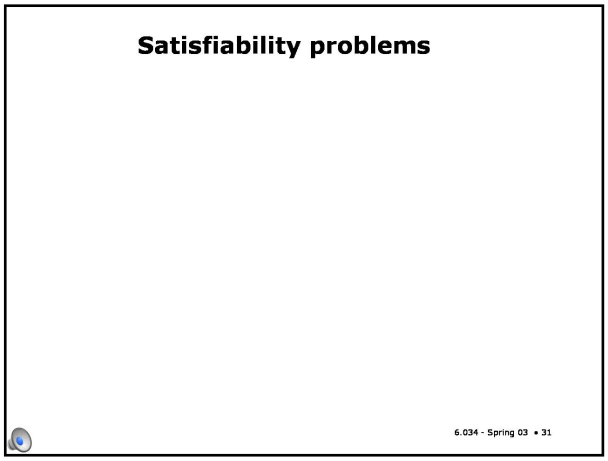
Slide 7.2.29
We could try to solve these problems using the brute-force method of enumerating all possible interpretations, then looking for one that makes the sentence true.

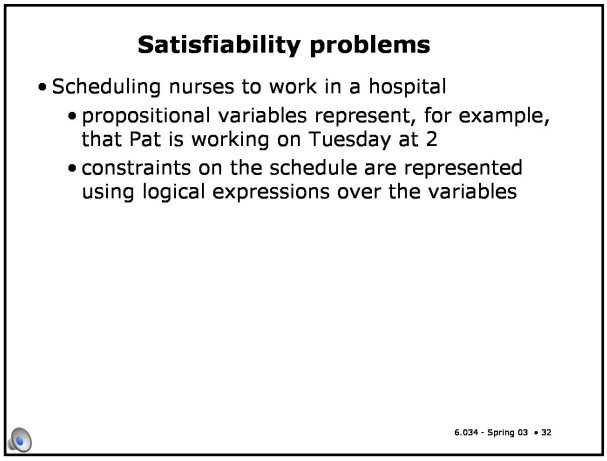




Slide 7.2.30
Better would be to use methods from constraint satisfaction. There are a number of search algorithms that have been specially adapted to solving satisfiability problems as quickly as possible, using combinations of backtracking, constraint propagation, and variable ordering.

Slide 7.2.31
There are lots of satisfiability problems in the real world. They end up being expressed essentially as lists of boolean logic expressions, where you're trying to find some assignment of values to variables that makes the sentence true.

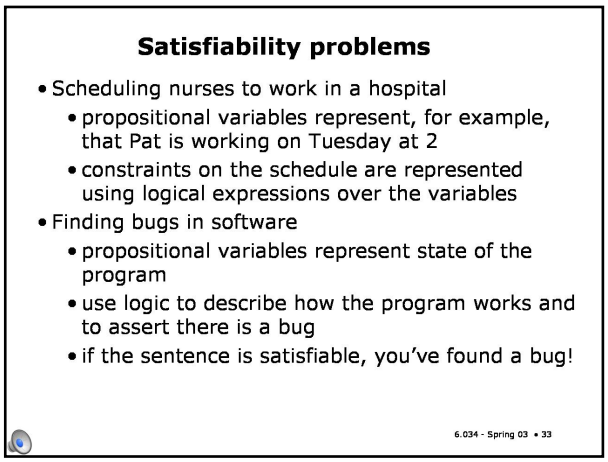




Slide 7.2.32
One example is scheduling nurses to work shifts in a hospital. Different people have different constraints, some don't want to work at night, no individual can work more than this many hours out of that many hours, these two people don't want to be on the same shift, you have to have at least this many per shift and so on. So you can often describe a setting like that as a bunch of constraints on a set of variables.

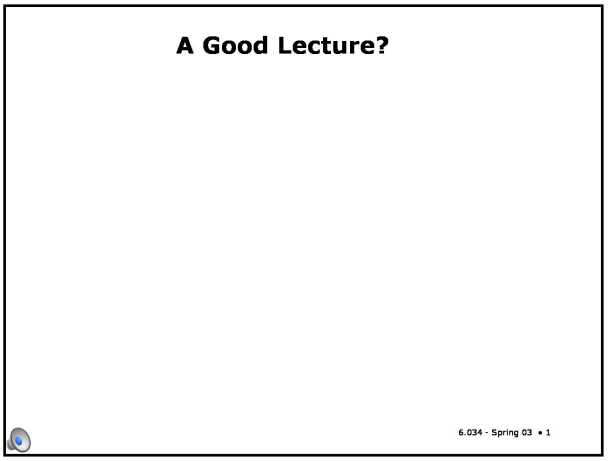
Slide 7.2.33
There's an interesting application of satisfiability that's going on here at MIT in the Lab for Computer Science. Professor Daniel Jackson's interested in trying to find bugs in programs. That's a good thing to do, but (as you know!) it's hard for humans to do reliably, so he wants to get the computer to do it automatically.

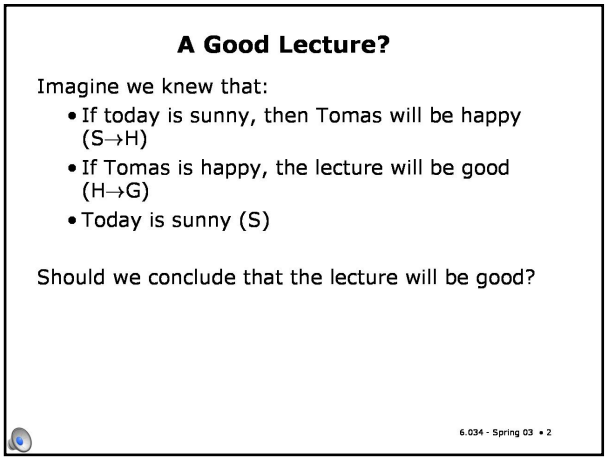
One way to do it is to essentially make a small example instance of a program. So an example of a kind of program that he might want to try to find a bug in would be an air traffic controller. The air traffic controller has rules that specify how it works. So you could write down the logical specification of how the air traffic control protocol works, and then you could write down another sentence that says, "and there are two airplanes on the same runway at the same time." And then you could see if there is a satisfying assignment; whether there is a configuration of airplanes and things that actually satisfies the specifications of the air traffic control protocol and also has two airplanes on the same runway at the same time. And if you can find one -- if that whole sentence is satisfiable, then you have a problem in your air traffic control protocol.



6.034 Notes: Section 7.3

Slide 7.3.1
One reason for writing down logical descriptions of situations is that they will allow us to draw conclusions about other aspects of the situation we've described.

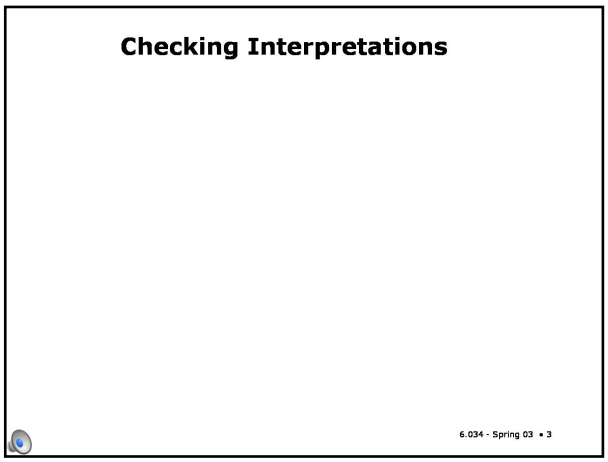


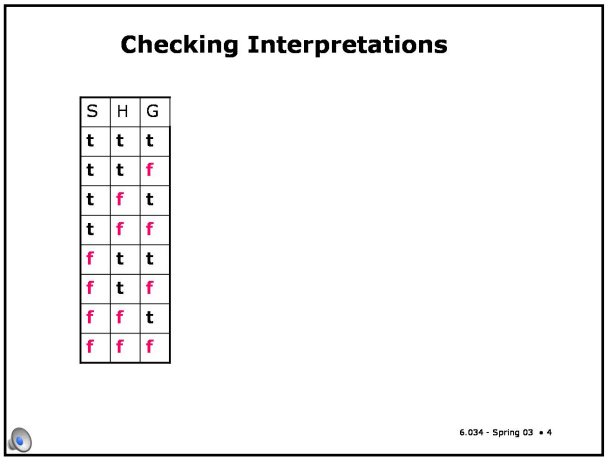


Slide 7.3.2
Imagine that we knew the following things to be true: If today is sunny, Tomas will be happy; if Tomas is happy, the lecture will be good; and today is sunny.

Does this mean that the lecture will be good?

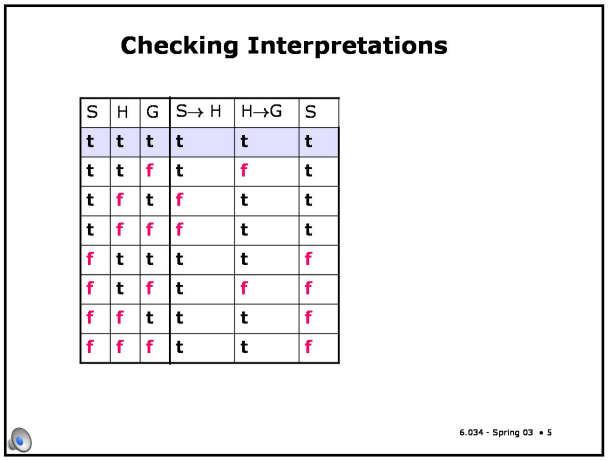
Slide 7.3.3
One way to think about this is to start by figuring out what set of interpretations make our original sentences true. Then, if G is true in all those interpretations, it must be okay to conclude it from the sentences we started out with (sometimes called our knowledge base).

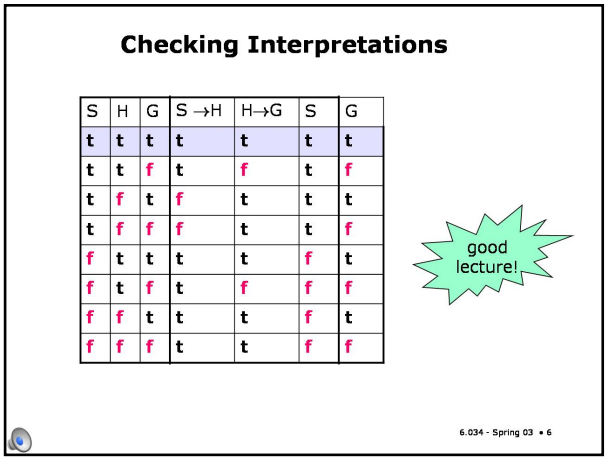




Slide 7.3.4
In a universe with only three variables, there are 8 possible interpretations in total.

Slide 7.3.5
Only one of these interpretations makes all the sentences in our knowledge base true: S = true, H = true, G = true.

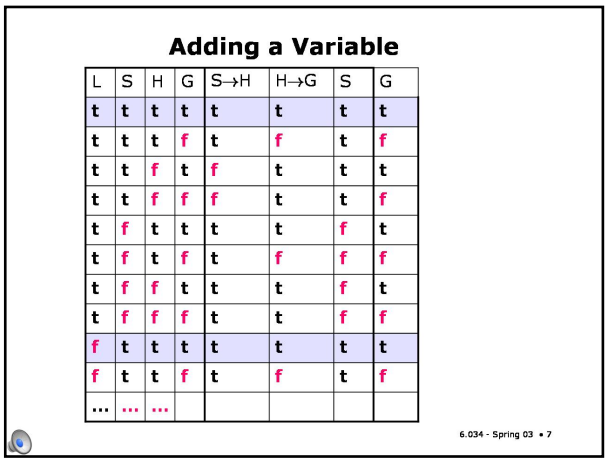


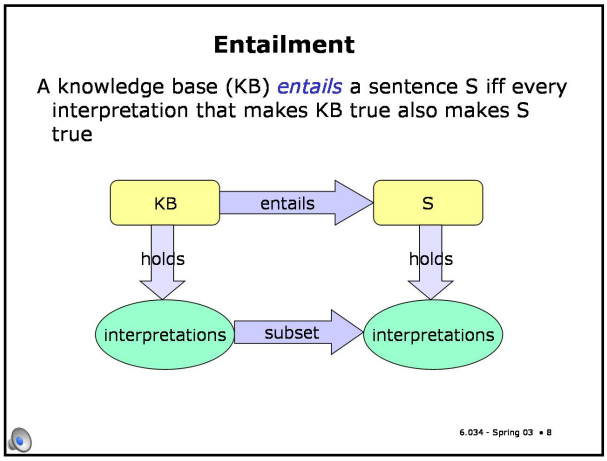


Slide 7.3.6
And it's easy enough to check that G is true in that interpretation, so it seems like it must be reasonable to draw the conclusion that the lecture will be good. (Good thing!).

Slide 7.3.7
If we added another variable to our domain, say whether Leslie is happy (L), then we'd have two interpretations that satisfy the KB: S = true, H = true, G = true, L = true; and S = true, H = true, G = true, L = false.

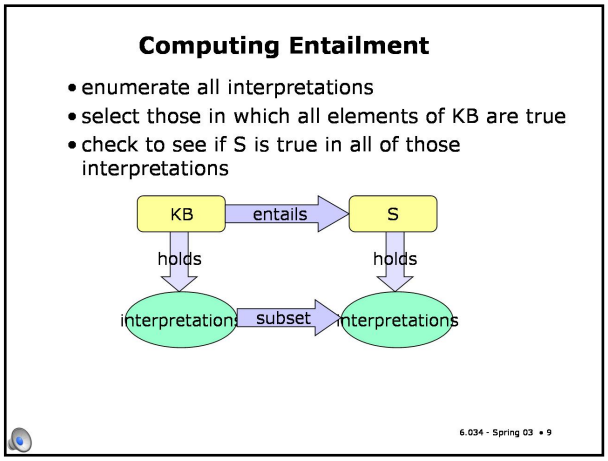
G is true in both of these interpretations, so, again, if the KB is true, then G must also be true.

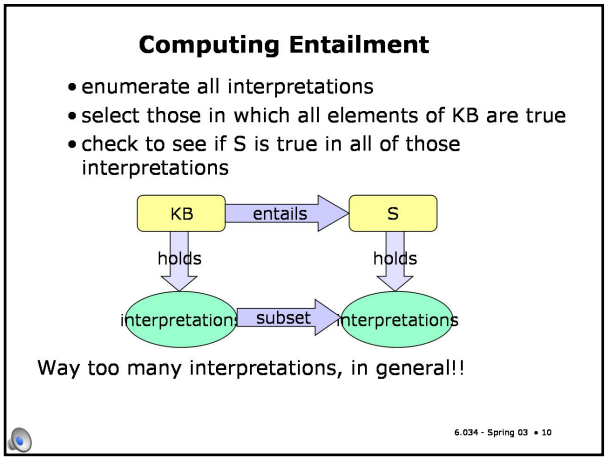




Slide 7.3.8
There is a general idea called "entailment" that signifies a relationship between a knowledge base and another sentence. If whenever the KB is true, the conclusion has to be true (that is, if every interpretation that satisfies the KB also satisfies the conclusion), we'll say that the KB "entails" the conclusion. You can think of entailment as something like "follows from", or "it's okay to conclude from".

Slide 7.3.9
The method of enumerating all the interpretations that satisfy the KB, and then checking to see if the conclusion is true in all of them is a correct way to test entailment.

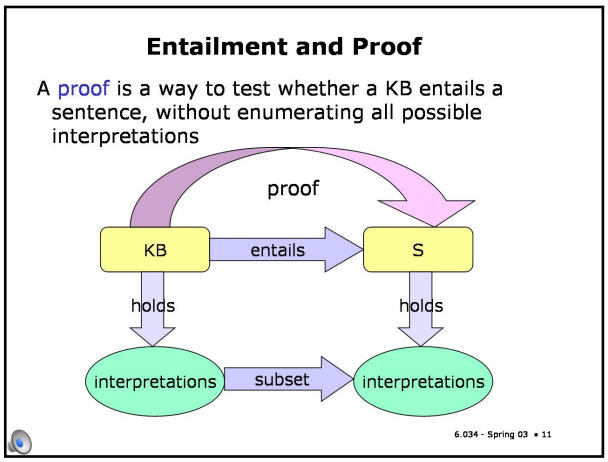


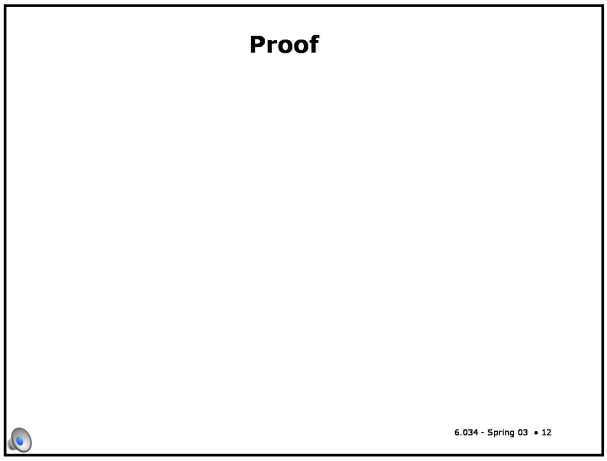


Slide 7.3.10
But now, what if we were to add 6 more propositional variables to our domain? Then we'd have 2^10 = 1024 interpretations to check, which is way too much work to do (and, in the first order case, we'll find that we might have infinitely many intepretations, which is definitely too much work to enumerate!!).

Slide 7.3.11
So what we'd really like is a way to figure out whether a KB entails a conclusion without enumerating all of the possible interpretations.

A proof is a way to test whether a KB entails a sentence, without enumerating all possible interpretations. You can think of it as a kind of shortcut arrow that works directly with the syntactic representations of the KB and the conclusion, without going into the semantic world of interpretations.



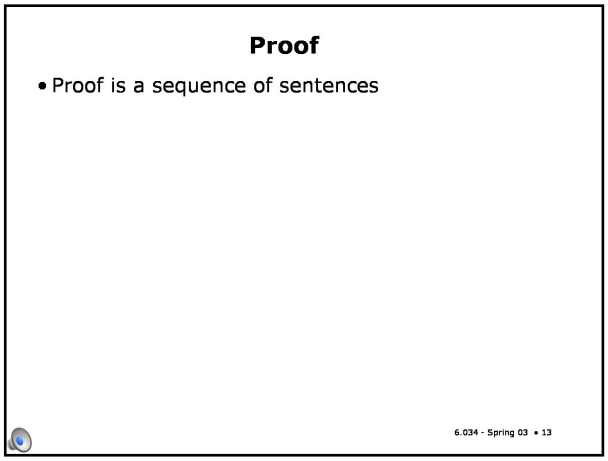


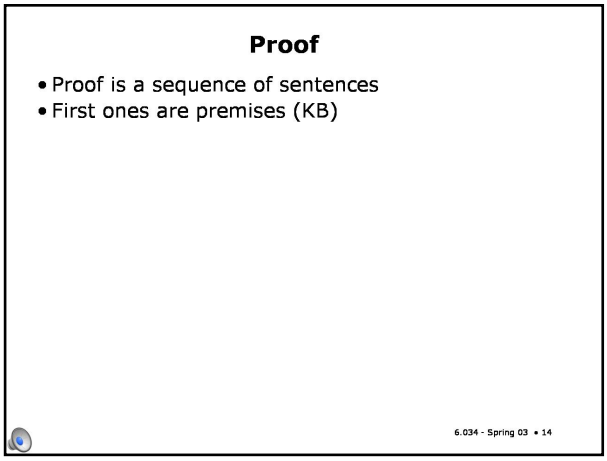
Slide 7.3.12
So what is a proof system? Well, presumably all of you have studied high-school geometry; that's often people's only exposure to formal proof. Remember that? You knew some things about the sides and angles of two triangles and then you applied the side-angle-side theorem to conclude -- at least people in American high schools were familiar with side-angle-side -- The side-angle-side theorem allowed you to conclude that the two triangles were similar, right?

That is formal proof. You've got some set of rules that you can apply. You've got some things written down on your page, and you grind through, applying the rules that you have to the things that are written down, to write some more stuff down until finally you've written down the things that you wanted to, and then you get to declare victory. That's a proof. There are (at least) two styles of proof system; we're going to talk about one briefly here and then go on to the other one at some length in the next two sections.

Natural deduction refers to a set of proof systems that are very similar to the kind of system you used in high-school geometry. We'll talk a little bit about natural deduction just to give you a flavor of how it goes in propositional logic, but it's going to turn out that it's not very good as a general strategy for computers. It's a proof system that humans like, and then we'll talk about a proof system that computers like, to the extent that computers can like anything.

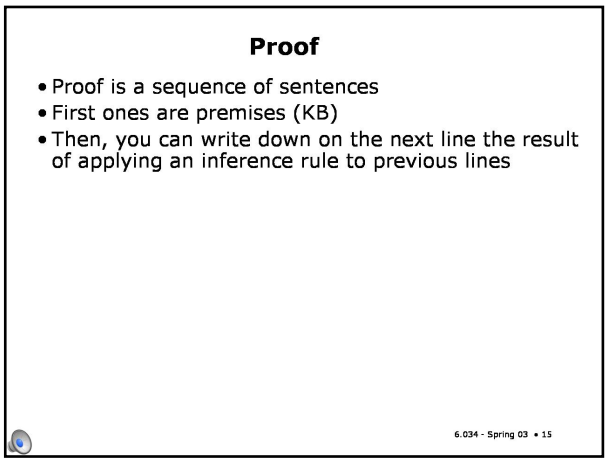
Slide 7.3.13
A proof is a sequence of sentences. This is going to be true in almost all proof systems.

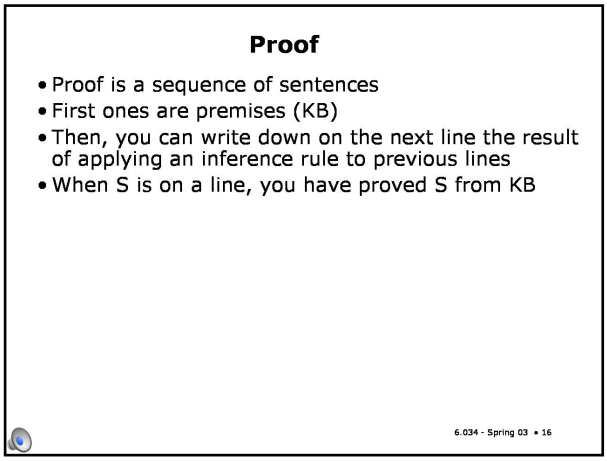




Slide 7.3.14
First we'll list the premises. These are the sentences in your knowledge base. The things that you know to start out with. You're allowed to write those down on your page. Sometimes they're called the "givens." You can put the givens down.

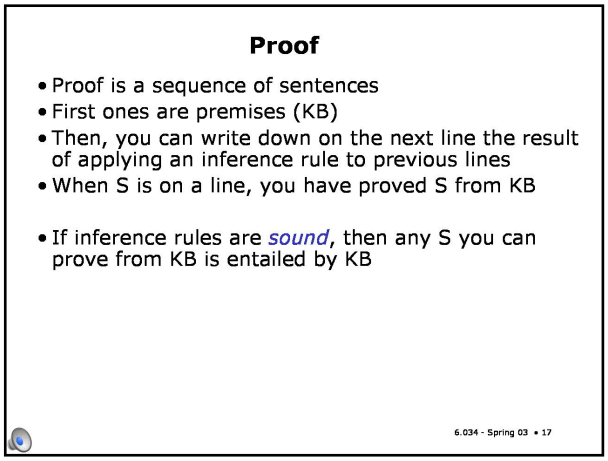
Slide 7.3.15
Then, you can write down on a new line of your proof the results of applying an inference rule to the previous lines.

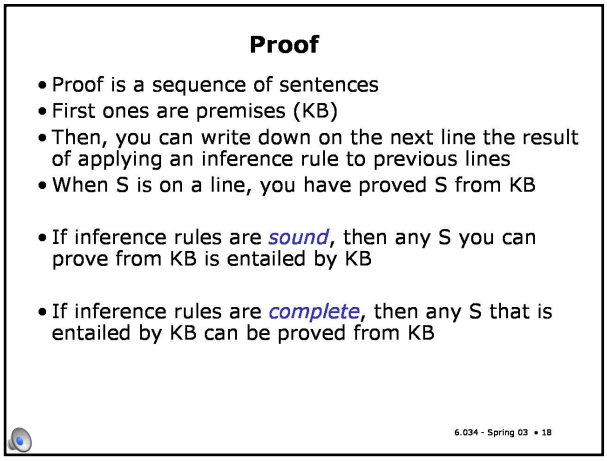




Slide 7.3.16
Then, when a sentence S is on some line, you have proved S from KB.

Slide 7.3.17
If your inference rules are **sound**, then any S you can prove from KB is, in fact, entailed by KB. That is, it's legitimate to draw the conclusion S from the assumptions in KB.

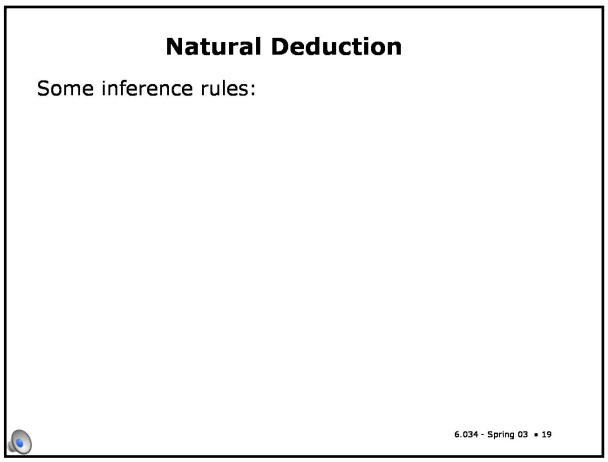


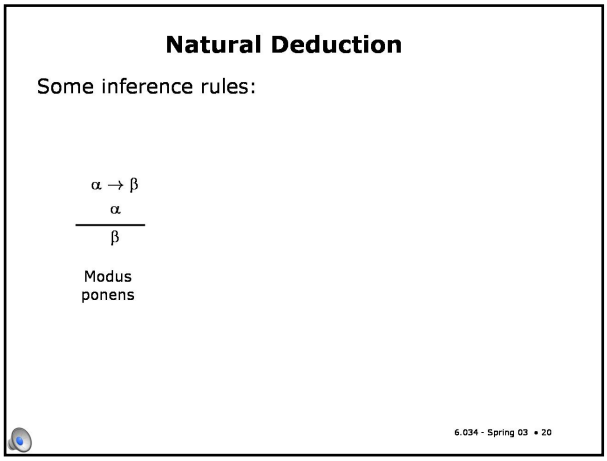


Slide 7.3.18
If your rules are **complete**, then you can use KB to prove any S that is entailed by the KB. That is, you can prove any legitimate conclusion.

Wouldn't it be great if you were sound and complete derivers of answers to problems? You'd always get an answer and it would always be right!

Slide 7.3.19
So let's look at inference rules, and learn how they work by example. We'll look at natural-deduction rules first, because they're easiest to understand.

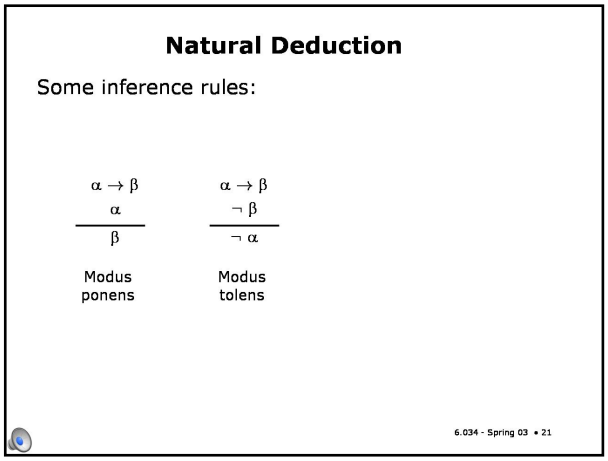


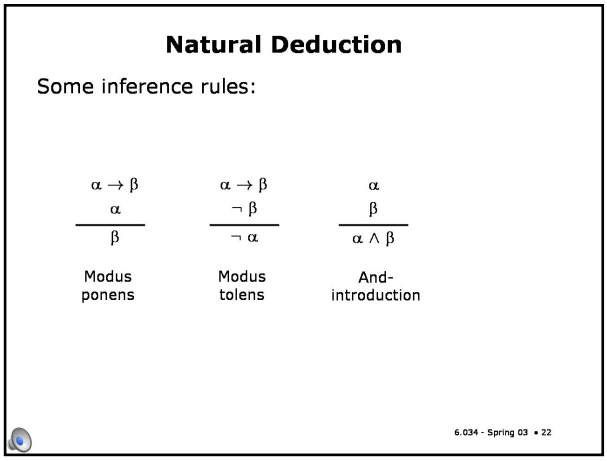


Slide 7.3.20
Here's a famous one (first written down by Aristotle); it has the great Latin name, "modus ponens", which means "affirming method".

It says that if you have "Alpha implies Beta" written down somewhere on your page, and you have Alpha written down somewhere on your page, then you can write beta down on a new line. (Alpha and Beta here are metavariables, like Phi and Psi, ranging over whole complicated sentences). It's important to remember that inference rules are just about ink on paper, or bits on your computer screen. They're not about anything in the world. Proof is just about writing stuff on a page, just syntax. But if you're careful in your proof rules and they're all sound, then at the end when you have some bit of syntax written down on your page, you can go back via the interpretation to some semantics. So you start out by writing down some facts about the world formally as your knowledge base. You do stuff with ink and paper for a while and now you have some other symbols written down on your page. You can go look them up in the world and say, "Oh, I see. That's what they mean."

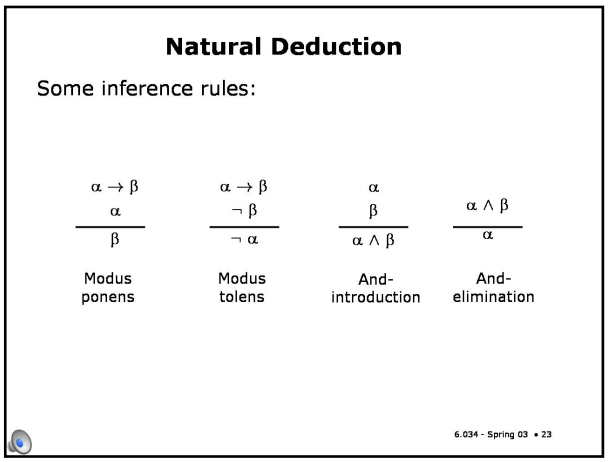
Slide 7.3.21
Here's another inference rule. "Modus tollens" (denying method) says that, from "alpha implies beta" and "not beta" you can conclude "not alpha".

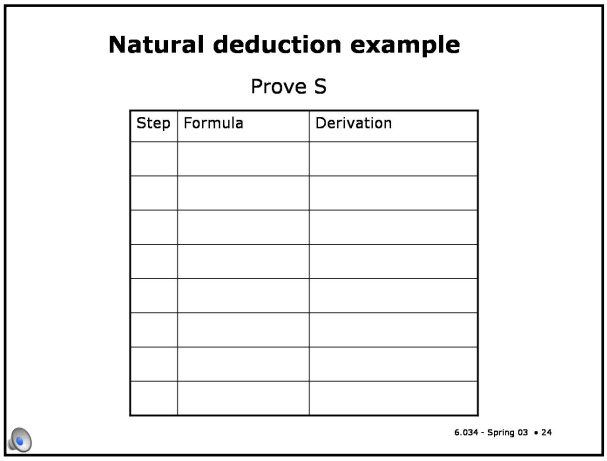




Slide 7.3.22
And-introduction say that from "Alpha" and from "Beta" you can conclude "Alpha and Beta". That seems pretty obvious.

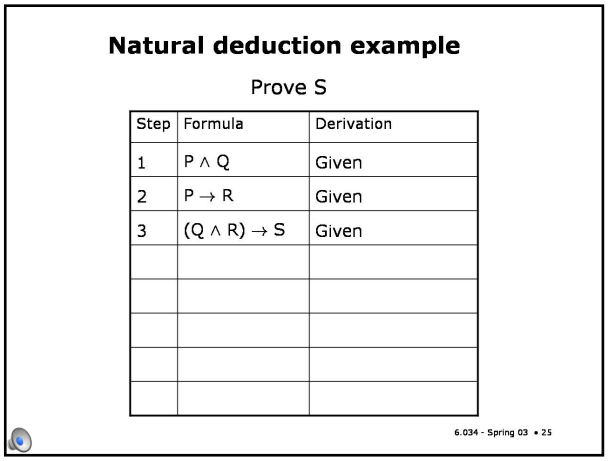
Slide 7.3.23
Conversely, and-elimination says that from "Alpha and Beta" you can conclude "Alpha".

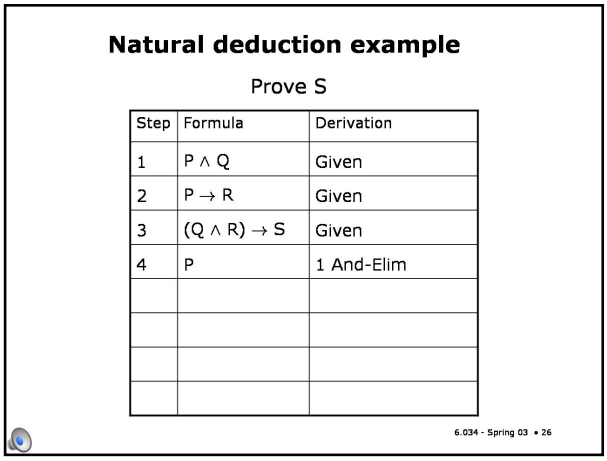




Slide 7.3.24
Now let's do a sample proof just to get the idea of how it works. Pretend you're back in high school

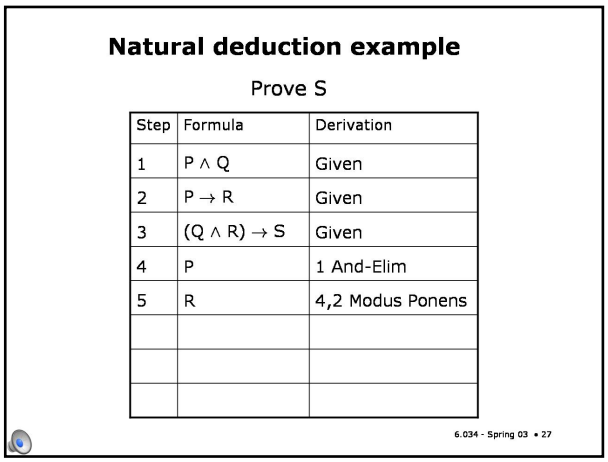
Slide 7.3.25
We'll start with 3 sentences in our knowledge base, and we'll write them on the first three lines of our proof: (P and Q), (P implies R), and (Q and R imply S).

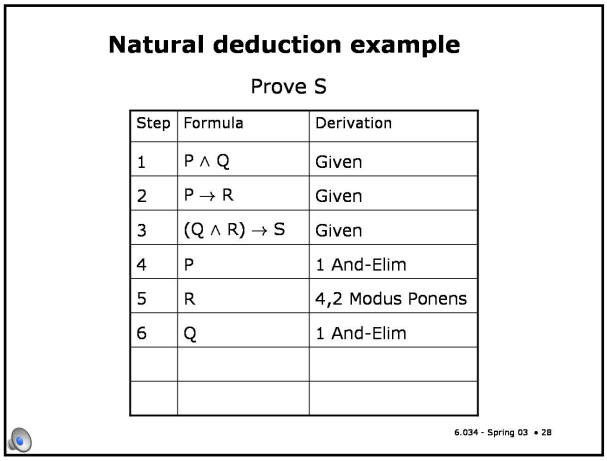




Slide 7.3.26
From line 1, using the and-elimination rule, we can conclude P, and write it down on line 4 (together with a reminder of how we derived it).

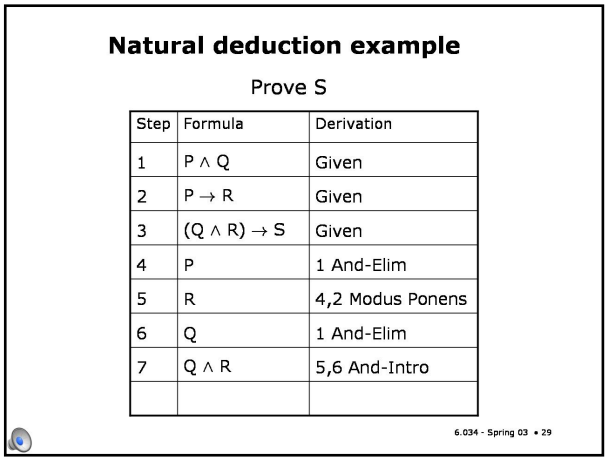
Slide 7.3.27
From lines 4 and 2, using modus ponens, we can conclude R.

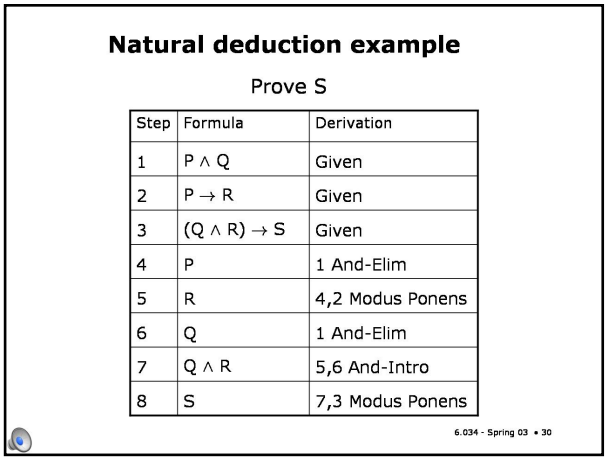




Slide 7.3.28
From line 1, we can use and-elimination to get Q.

Slide 7.3.29
From lines 5 and 6, we can use and-introduction to get (Q and R).

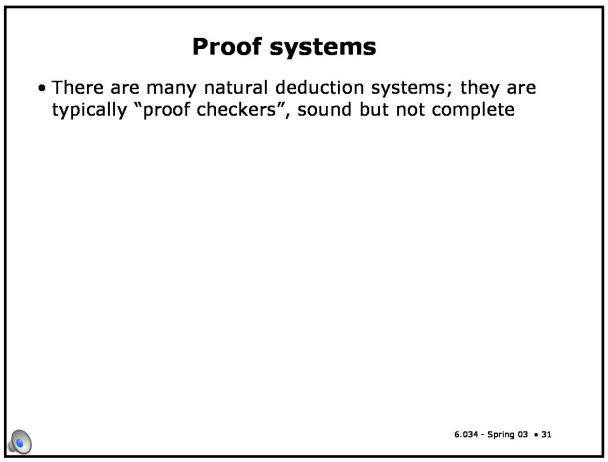


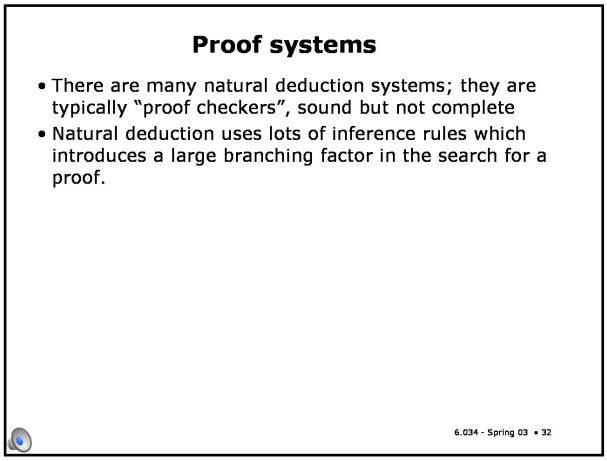


Slide 7.3.30
Finally, from lines 7 and 3, we can use modus ponens to get S. Whew! We did it!

Slide 7.3.31
The process of formal proof seems pretty mechanical. So why can't computers do it?

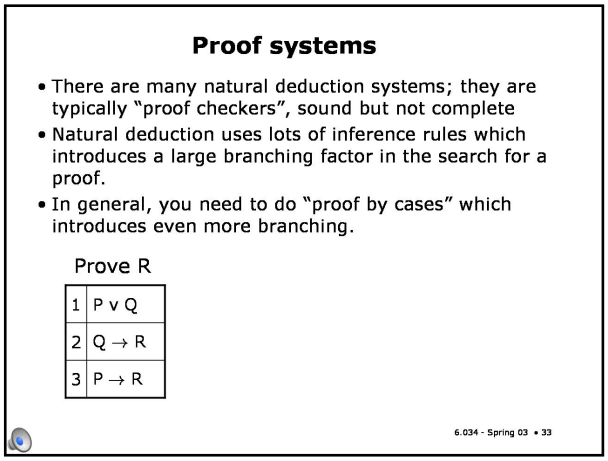
They can. For natural deduction systems, there are a lot of "proof checkers", in which you tell the system what conclusion it should try to draw from what premises. They're always sound, but nowhere near complete. You typically have to ask them to do the proof in baby steps, if you're trying to prove anything at all interesting.

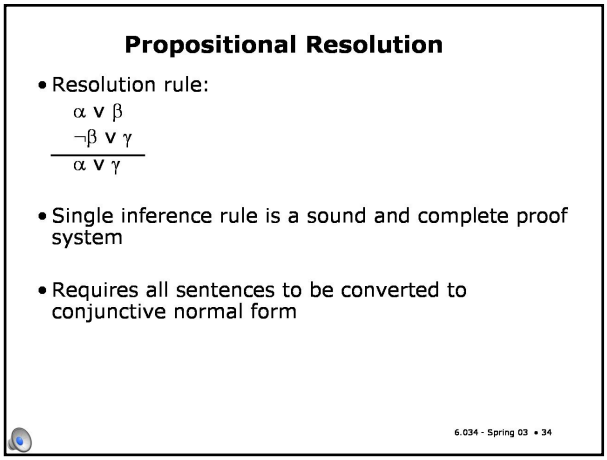




Slide 7.3.32
Part of the problem is that they have a lot of inference rules, which introduces a very big branching factor in the search for proofs.

Slide 7.3.33
Another big problem is the need to do "proof by cases". What if you wanted to prove R from (P or Q), (Q implies R), and (P implies R)? You have to do it by first assuming that P is true and proving R, then assuming Q is true and proving R. And then finally applying a rule that allows you to conclude that R follows no matter what. This kind of proof by cases introduces another large amount of branching in the space.



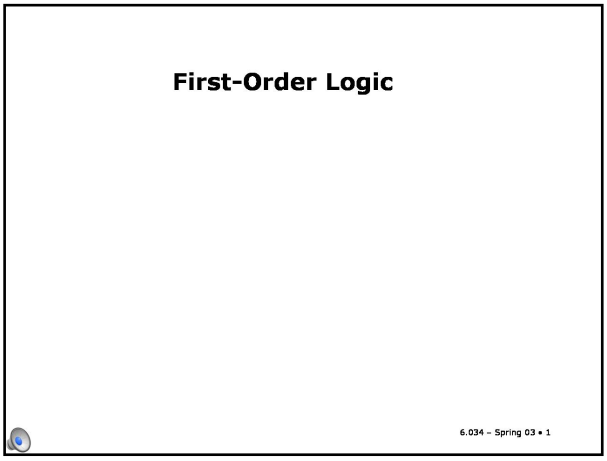


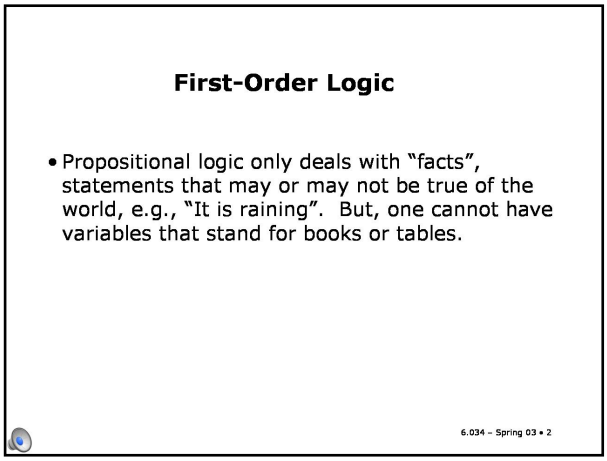
Slide 7.3.34
An alternative is resolution, a single inference rule that is sound and complete, all by itself. It's not very intuitive for humans to use, but it's great for computers.

Resolution requires all sentences to be first written in a special form. So the next section will investigate that special form, and then we'll return to resolution.

6.034 Notes: Section 7.4

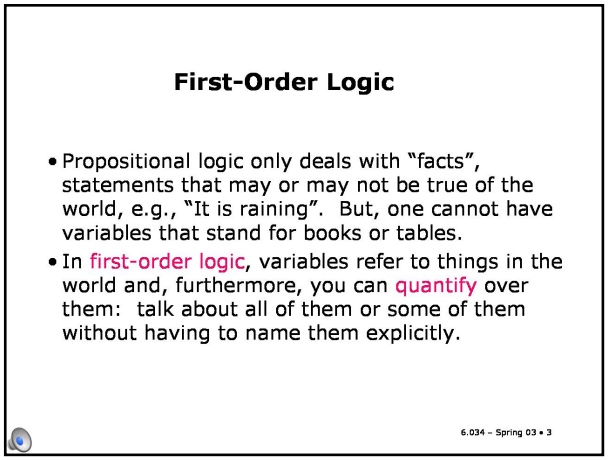
Slide 7.4.1
Now we're going to start talking about first-order logic, which extends propositional logic so that we can talk about *things*.

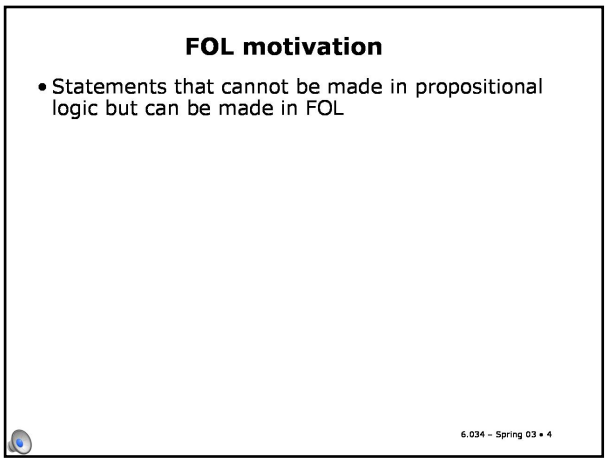




Slide 7.4.2
In propositional logic, all we had were variables that stood, not for things in the world or even quantities, but just facts, Boolean statements that might or might not be true about the world, like whether it's raining, or greater than 67 degrees; but you couldn't have variables that stood for tables or books, or the temperature, or anything like that. And as it turns out, that's an enormously limiting kind of representation.

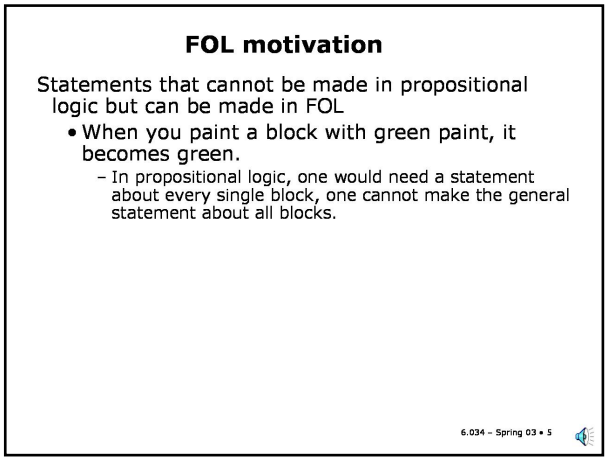
Slide 7.4.3
In first-order logic, variables refer to things in the world and you can quantify over them. That is, you can talk about all or some of them without having to name them explicitly.

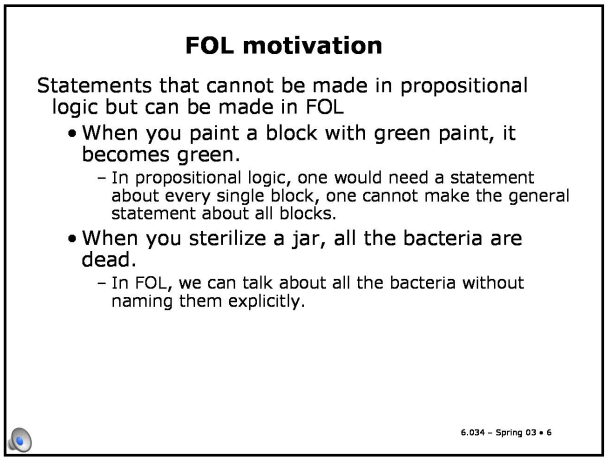




Slide 7.4.4
There are lots of examples that show how propositional logic is inadequate to characterize even moderately complex domains. Here are some more examples of the kinds of things that you can say in first-order logic, but not in propositional logic.

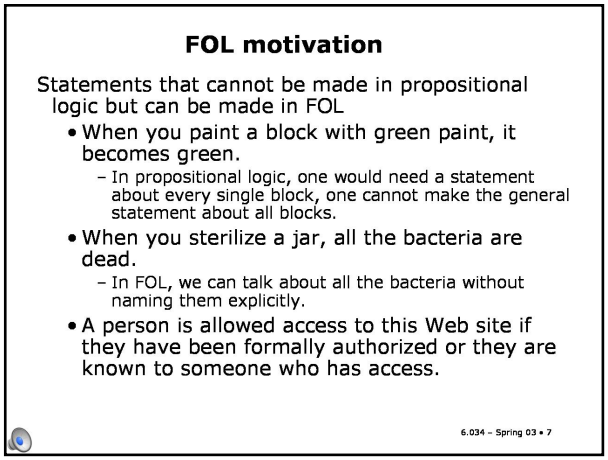
Slide 7.4.5
"When you paint the block, it becomes green." You might have a proposition for every single aspect of the situation, like "if this block is black and I paint it, it becomes green" and "if that block is red and I paint it, it becomes green" and "if block #5 is green and I paint it, it becomes green". But you'd have to have one of those propositions for every single initial block color, or every single block, or every single object (if you have non-blocks, too) in the world. You couldn't say that, as a general fact, "after you paint something if becomes green."

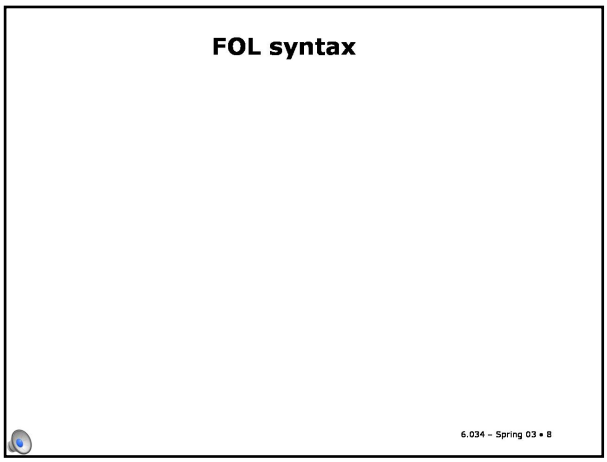




Slide 7.4.6
Let's say you want to talk about what happens when you sterilize a jar. It kills all the bacteria in the jar. Now, you don't want to have to name all the bacteria; to have to say, bacterium 57 is dead, and bacterium 93 is dead. Each one of those guys is dead. All the bacteria are dead now. So you'd like to have a way not only to talk about things in the world, but to talk about all of them, or some of them, without naming any of them explicitly.

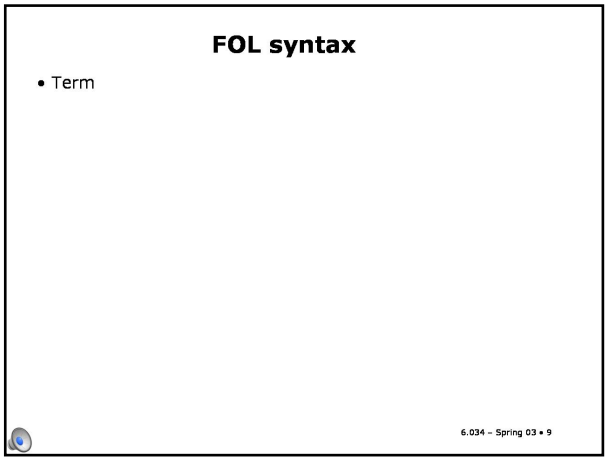
Slide 7.4.7
In the context of providing flexible computer security, you might want to prove or try to understand whether someone should be allowed access to a web site. And you could say: a person should have access to this web site if they've been personally, formally authorized to use this web site or if they are known to someone who has access to the web site. So you could write a general rule that says that and then some other system or this system could try to prove that you should have access to the web site. In this case, what that would mean would be going to look for a chain of people that are authorized or known to one another that bottoms out in somebody who's known to this web site.

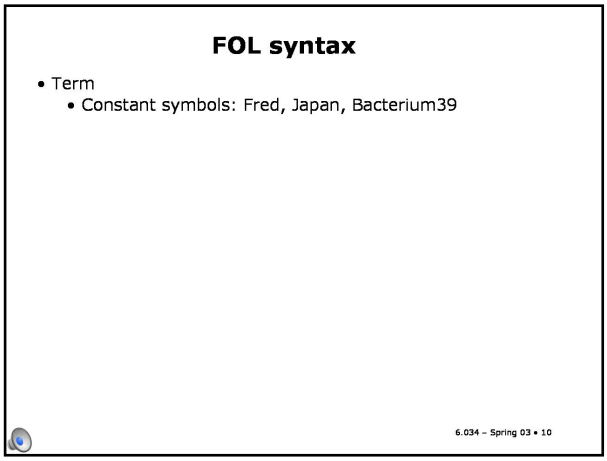




Slide 7.4.8
First-order logic lets us talk about things in the world. It's a logic like propositional logic, but somewhat richer and more complex. We'll go through the material in the same way that we did propositional logic: we'll start with syntax and semantics, and then do some practice with writing down statements in first-order logic.

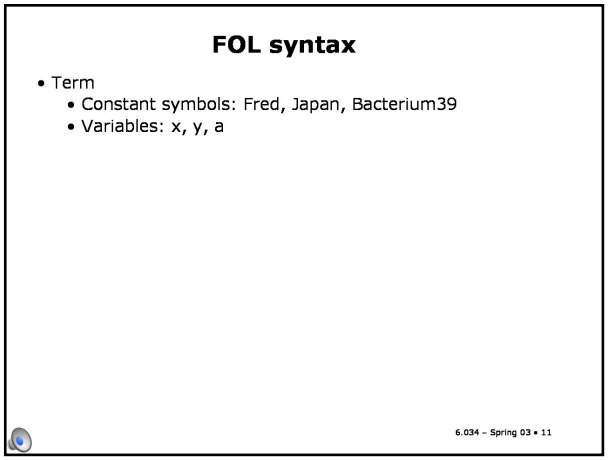
Slide 7.4.9
The big difference between propositional logic and first-order logic is that we can talk about things, and so there's a new kind of syntactic element called a *term*. And the term, as we'll see when we do the semantics, is a name for a thing. It's an expression that somehow names a thing in the world. There are three kinds of terms:

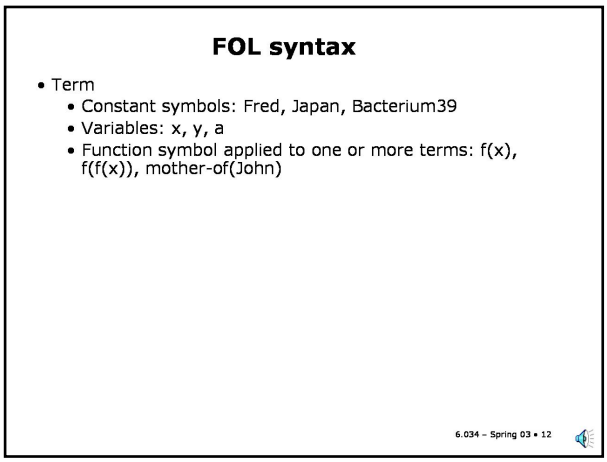




Slide 7.4.10
There are constant symbols. They are names like **Fred** or **Japan** or **Bacterium39**. Those are symbols that, in the context of an interpretation, name a particular thing.

Slide 7.4.11
Then there are variables, which are not really syntactically differentiated from constant symbols. We'll use capital letters to start constant symbols (think of them as proper names), and lower-case letters for term variables. (It's important to note, though, that this convention is not standard, and in some logic contexts, such as the programming language Prolog, they adopt the exact opposite convention).



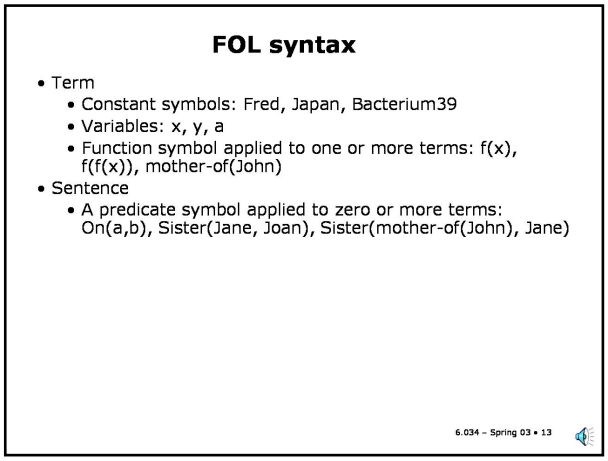


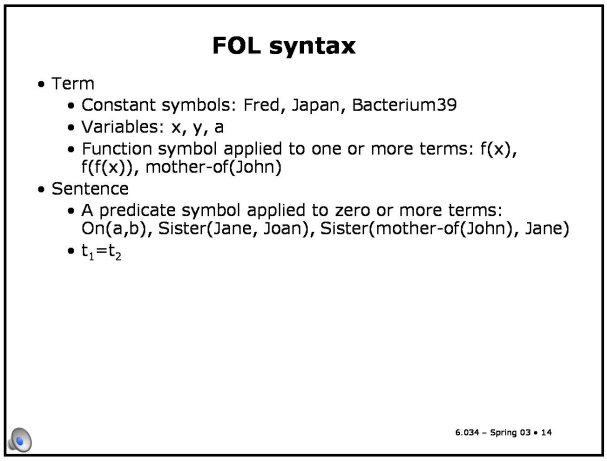
Slide 7.4.12
The last kind of term is a function symbol, applied to one or more terms. We'll use lower-case for function symbols as well. So another way to make a name for something is to say something like **f(x)**. If **f** is a function, you can give it a term and then **f(x)** names something. So, you might have **mother-of (John)** or **f(f(x))**. Note that a function with no terms would be a constant.

These three kinds of terms are our ways to name things in the world.

Slide 7.4.13
In propositional logic we had sentences. Now, in first-order logic it's a little bit more complicated, but not a lot. So what's a sentence? There's another kind of symbol called a predicate symbol. A predicate symbol is applied to zero or more terms. Predicate symbols stand for relations, so we might have things like **On(A,B)** or **Sister(Jane, Joan)**. **On** and **Sister** are predicate symbols; **a**, **b**, **Jane**, **Joan**, and **mother-of(John)** are terms.

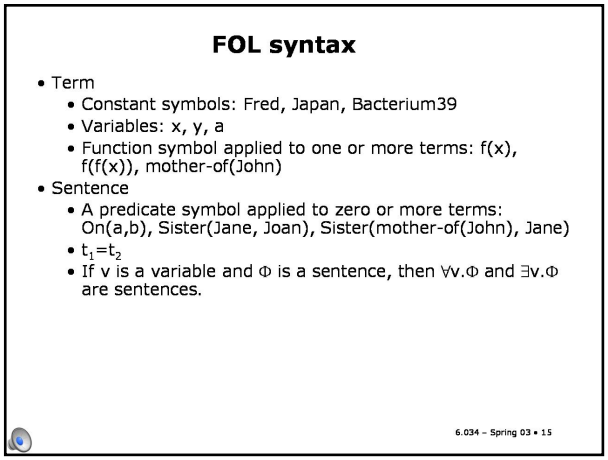
A predicate applied to zero terms is what's sometimes called a sentential variable or a propositional variable. It was our old kind of variable that we had before in propositional logic, like "it's-raining." It's a little bit of an artifice, but we'll take predicates with no arguments to be variables that have values true or false.

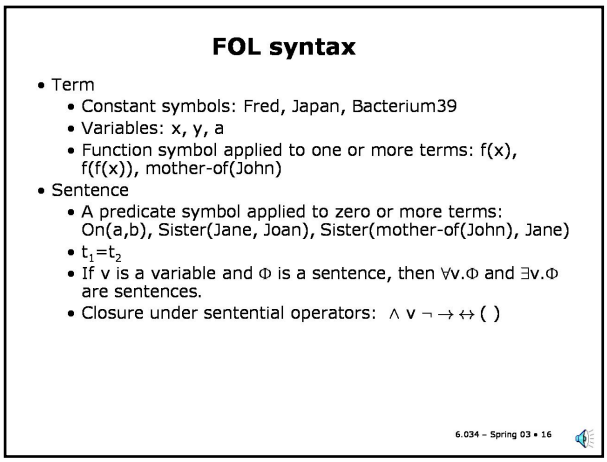




Slide 7.4.14
A sentence can also be of the form $\mathbf{t_1 = t_2}$. We're going to have one special predicate called equality. You can say this thing equals that thing, written term, equal-sign, term.

Slide 7.4.15
There are two more new constructs. If **v** is a variable and **Phi** is a sentence then **(upside-down-A v . phi)**, and **(backwards-E v . phi)** are sentences. You've probably seen these symbols before informally as "for all" and "there exists", and that's what they're going to mean for us, too.



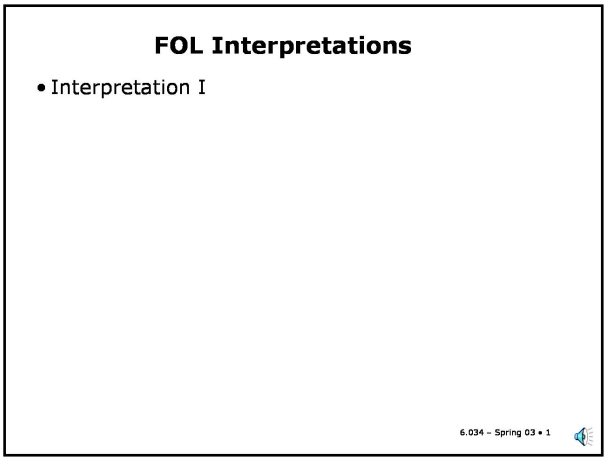


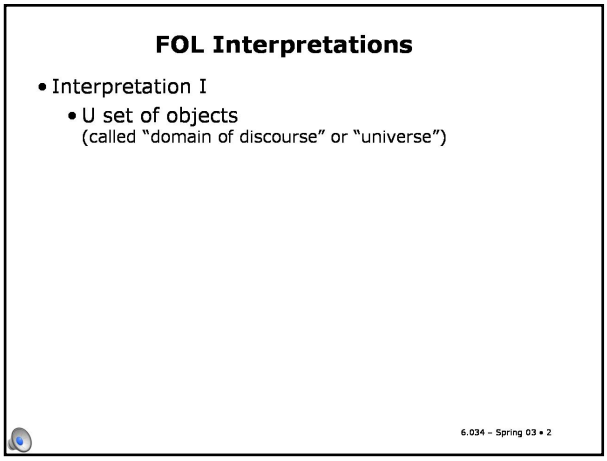
Slide 7.4.16
Finally we have closure under the sentential operators that we had before, so you can make complex sentences out of other sentences using and, or, not, implies, equivalence (also called biconditional), and parentheses, just as before in propositional logic. All that basic connective structure is still the same, but the things that we can say on either side have gotten a little bit more complicated.

All right, that's our syntax. That's what we get to write down on our page.

6.034 Notes: Section 7.5

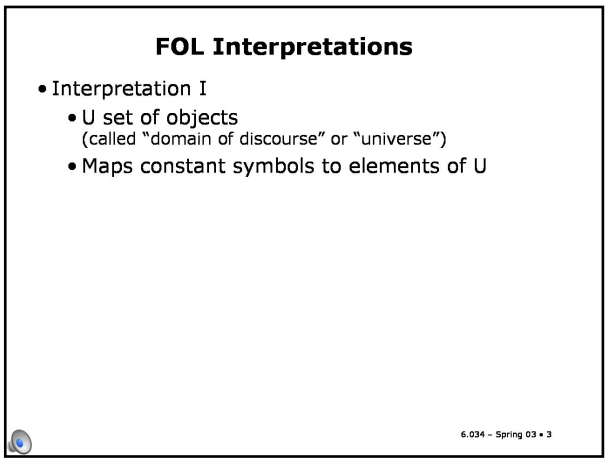
Slide 7.5.1
We're going to do the semantics informally. This isn't really going to look informal to you, but compared to the sorts of things that logicians write down, it's pretty informal. In propositional logic, an interpretation (**I**) is an assignment of truth values to sentential variables. Now an interpretation's going to be something more complicated. An interpretation is made up of a set and three mappings.

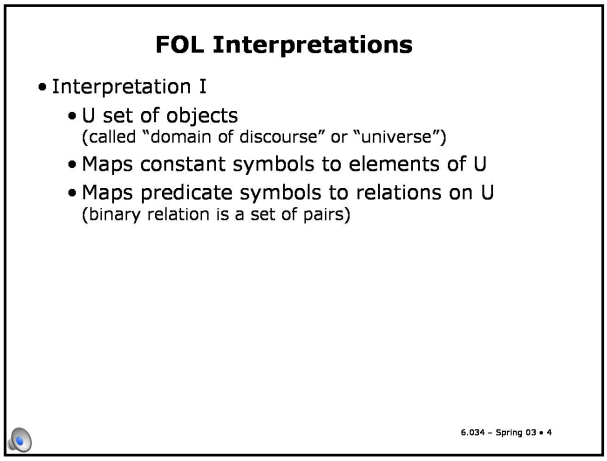




Slide 7.5.2
The set is the universe, **U**, which is a set of objects. So what's an object? Well, really, it could be this chair and that chair and these pieces of chalk or it could be all of you guys or it could be some trees out there, or it could be rather more abstract objects like meetings or points in time or numbers. An object could be anything you can think of, and the universe can be any set (finite or infinite) of objects. The universe is also sometimes called the "domain of discourse."

Slide 7.5.3
There's a mapping from constant symbols to elements of **U**, specifying how names are connected to objects in the world. So I might have the constant symbol, **Fred**, and I might have a particular person in the universe, and then the interpretation of the symbol **Fred** could be that person.

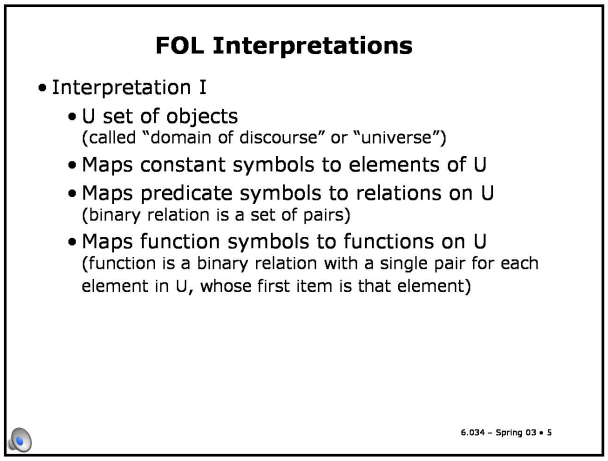


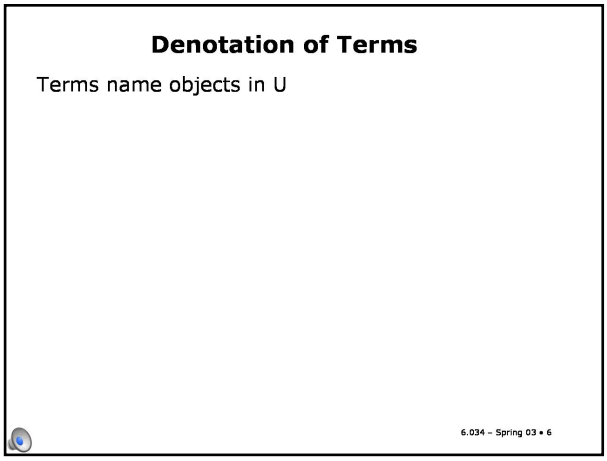


Slide 7.5.4
The next mapping is from predicate symbols to relations on **U**. An n-ary relation is a set of lists of n objects, saying which groups of things stand in that particular relation to one another. A binary relation is a set of pairs. So if I have a binary relation **brother-of** and **U** is a bunch of people, then the relation would be the set of all pairs of people such that the second is the brother of the first.

Slide 7.5.5
The last mapping is from function symbols to functions on **U**. Functions are a special kind of relation, in which, for any particular assignment of the first n-1 elements in each list, there is a single possible assignment of the last one. In the **brother-of** relation, there could be many pairs with the same first item and a different second item, but in a function, if you have the same first item then you have to have the same second item. So that means you just name the first item and then there's a unique thing that you get from applying the function. So it's OK for **mother-of** to be a function, discounting adoptions and other unusual situations. We will also, for now, assume that our functions are *total*, which means that there is an entry for every possible assignment of the first n-1 elements.

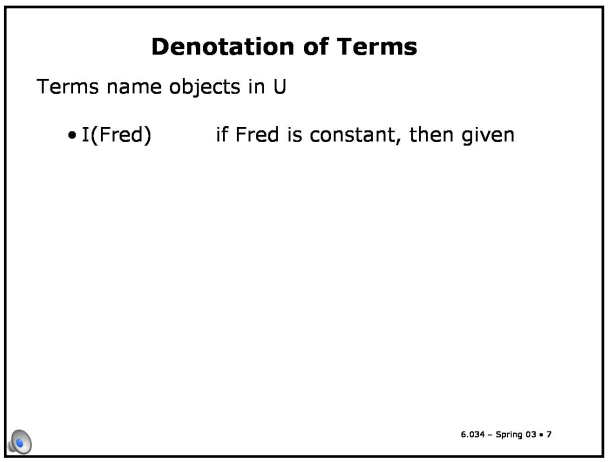
So, the last mapping is from function symbols to functions on the universe.

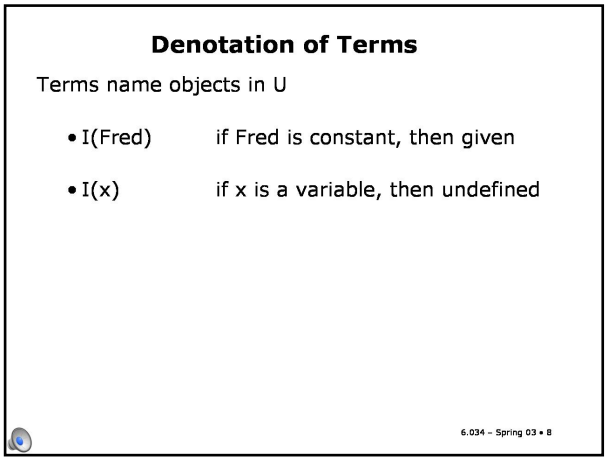




Slide 7.5.6
Before we can do the part of semantics that says what sentences are true in which interpretation, we have to talk about what terms mean. Terms name things, but we like to be fancy so we say a term denotes something, so we can talk about the denotation of a term, that is, the thing that a term names.

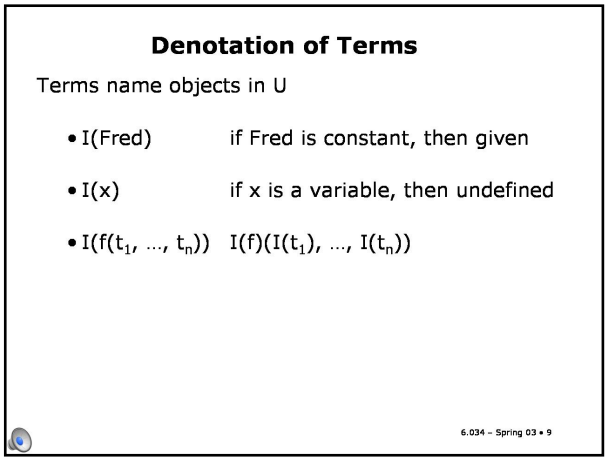
Slide 7.5.7
The denotations of constant symbols are given directly in the interpretation.

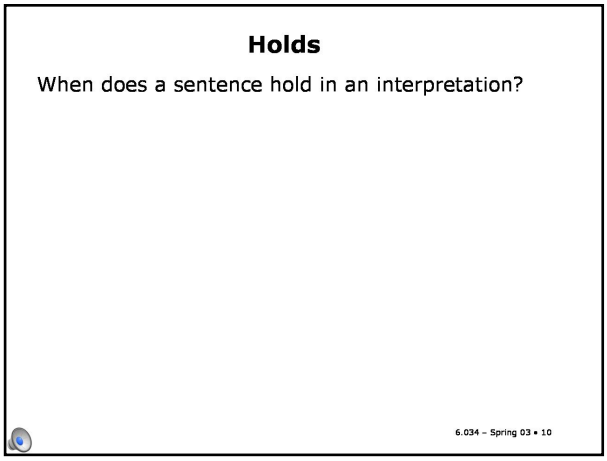




Slide 7.5.8
The denotation of a variable is undefined. What does x mean, if x is a variable? The answer is, "mu." It doesn't mean anything. That's a Zen joke. If you don't get it, don't worry about it.

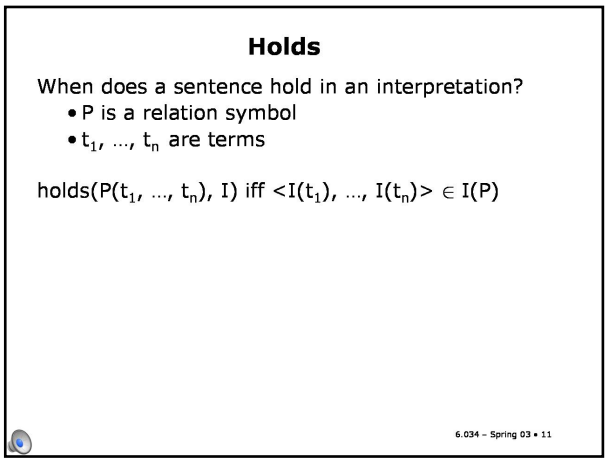
Slide 7.5.9
The denotation of a complex term is defined recursively. So, to find the interpretation of a function symbol applied to some terms, first you look up the function symbol in the interpretation and get a function. (Remember that the function symbol is a syntactic thing, ink on paper, but the function it denotes is an abstract mathematical object.) Then you find the interpretations of the component terms, which will be objects in **U**. Finally, you apply the function to the objects, yielding an object in **U**. And that object is the denotation of the complex term.

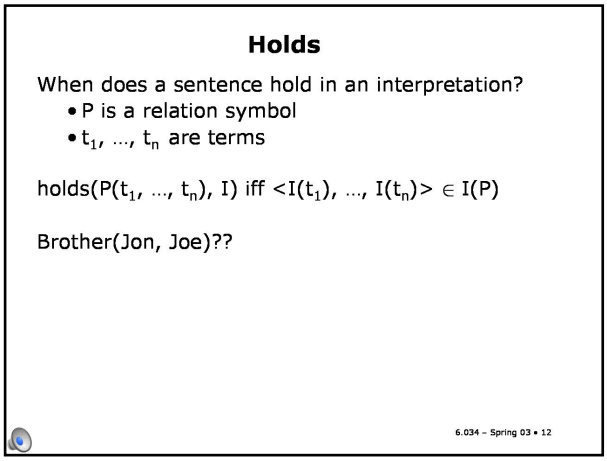




Slide 7.5.10
In the context of propositional logic, we looked at the rules of semantics, which told us how to determine whether a sentence was true in an interpretation. Now, in first-order logic, we'll add some semantic rules, for the new kinds of sentences we've introduced. One of our new kinds of sentences is a predicate symbol applied to a bunch of terms. That's a sentence, which is going to have a truth value, true or false.

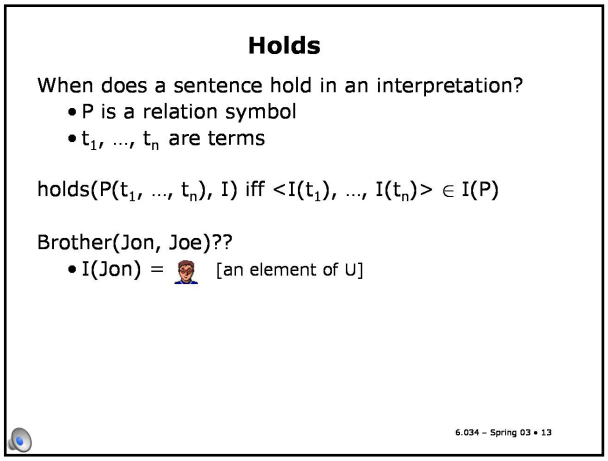
Slide 7.5.11
To figure out its truth value, we first use the denotation rules to find out which objects are named by each of the terms. Then, we look up the predicate symbol in the interpretation, which gives us a mathematical relation on **U**. Finally, we look to see if the list of objects named by the terms is a member of the relation. If so, the sentence is true in the given interpretation.

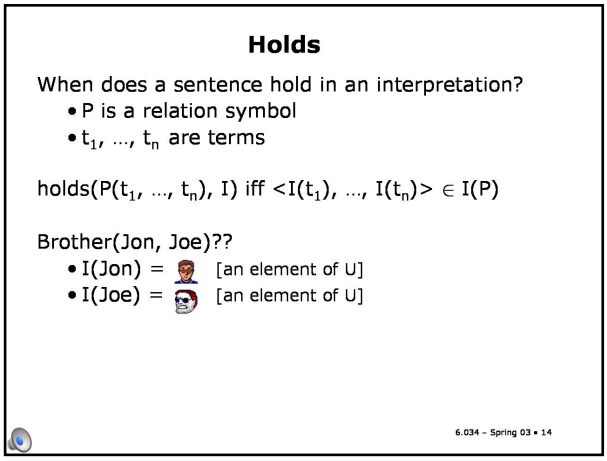




Slide 7.5.12
Let's look at an example. Imagine we want to determine whether the sentence **Brother(Jon,Joe)** is true in some interpretation.

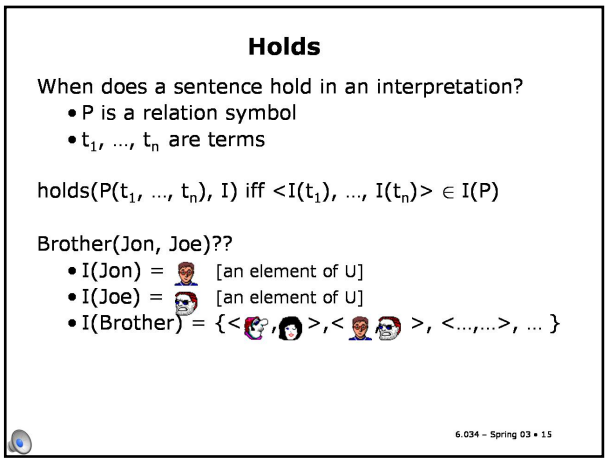
Slide 7.5.13
First, we look up the constant symbol **Jon** in the interpretation and find that it names this guy with glasses.

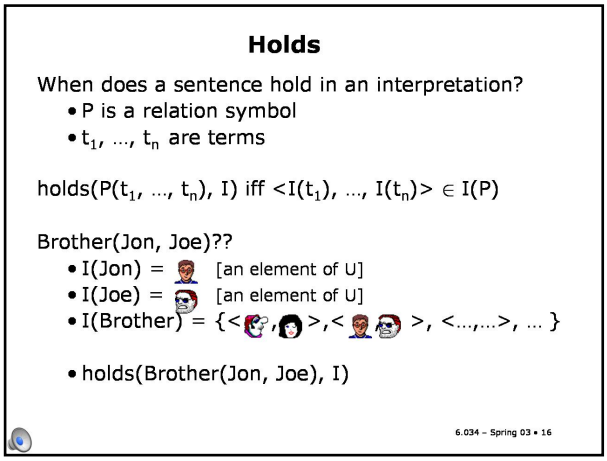




Slide 7.5.14
Then we look up **Joe** and find that it names this angry-looking guy.

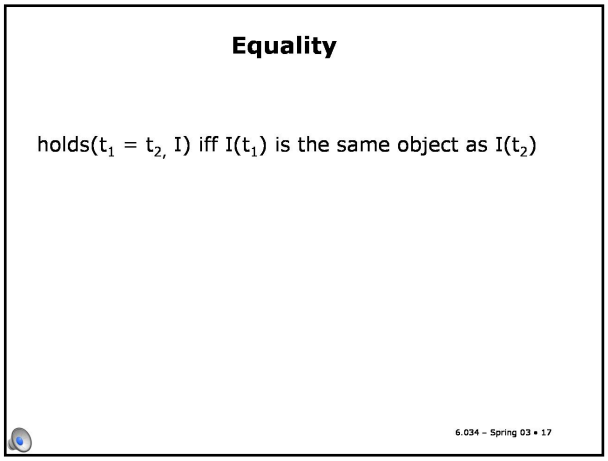
Slide 7.5.15
Now we look up the predicate symbol **Brother** and find that it denotes this complicated relation.

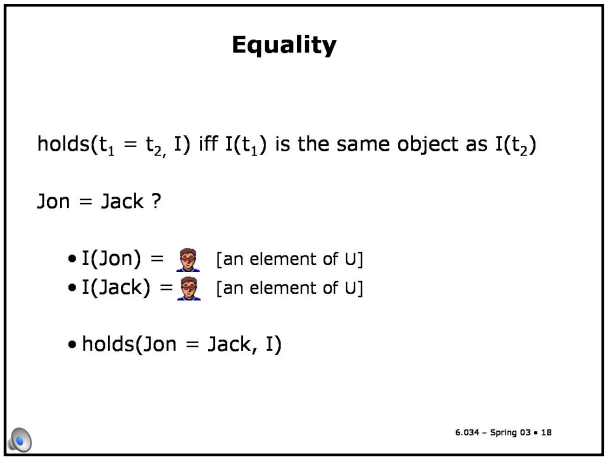




Slide 7.5.16
Finally, we look up the pair of the guy with glasses and the angry-looking guy, to see if they're in the relation. They are, so the sentence must be true in that interpretation. It's easy to think of lots of other interpretations in which it wouldn't be true (and lots of others in which it would).

Slide 7.5.17
Another new kind of sentence we introduced has the form **term$_1$ = term$_2$**. The semantics are pretty unsurprising: if the object denoted by **term$_1$** is the same as the object denoted by **term$_2$**, then the sentence holds.

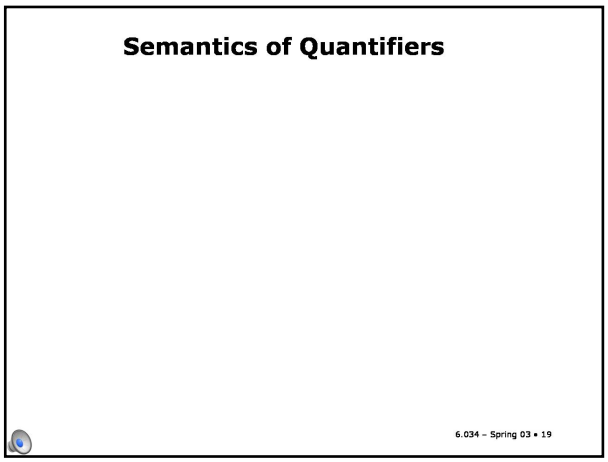


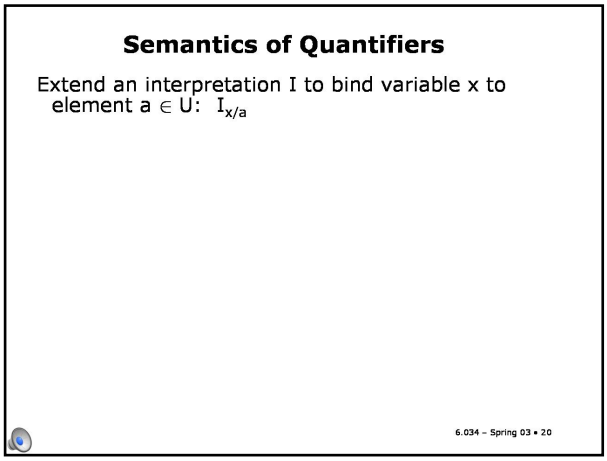


Slide 7.5.18
It's important to note that two different constant symbols can denote the same object in the universe; so this is not a test on equality of names.

We might have an interpretation that maps the symbols **Jon** and **Jack** both into the same guy. In that case, **Jon** = **Jack** holds in **I**.

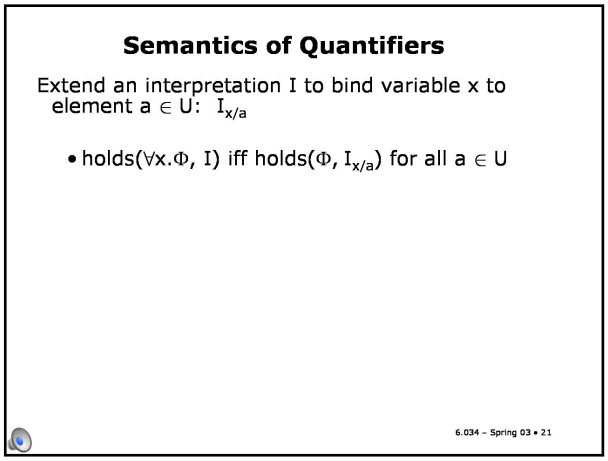
Slide 7.5.19
Now we have to figure out how to tell whether sentences with quantifiers in them are true.

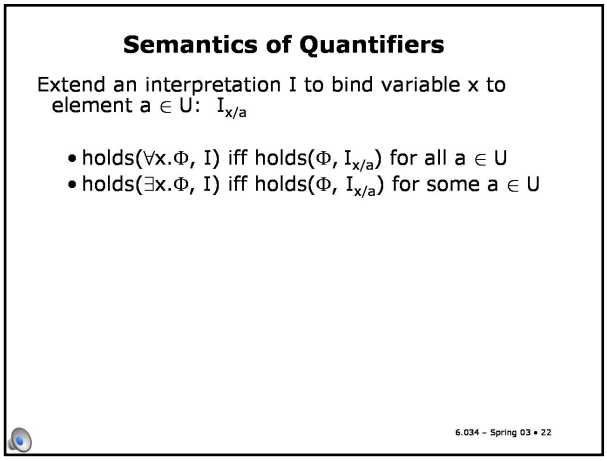




Slide 7.5.20
In order to talk about quantifiers we need the idea of extending an interpretation. We would like to be able to extend an interpretation to bind variable **x** to value **a**. We'll write that as **I** with **x** bound to **a**. Here, **x** is a variable and **a** is an object; an element of **U**. The idea is that, in order to understand whether a sentence that has variables in it is true or not, we have to make various temporary assignments to the variables and see what the truth value of the sentence is. Binding **x** to **a** is kind of like adding **x** as a constant symbol to **I**. It's kind of like temporarily binding a variable in a programming language.

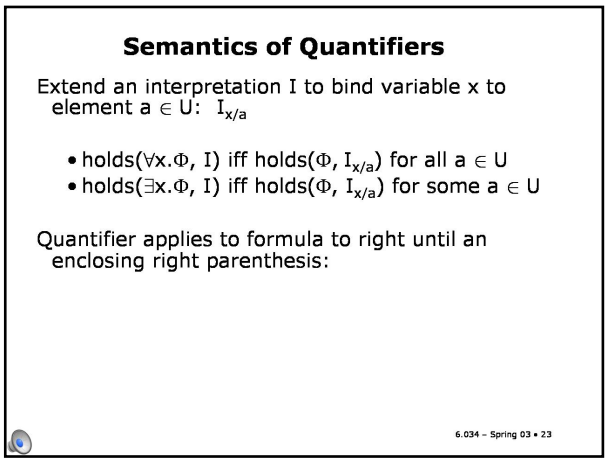
Slide 7.5.21
Now, how do we evaluate the truth under interpretation **I**, of the statement **for all x, Phi**? So how do we know if that's true? Well, it's true if and only if **Phi** is true if for every possible binding of variable **x** to thing in the world **a**. Okay? For every possible thing in the world that you could plug in for **x**, this statement's true. That's what it means to say **for all x, Phi**.

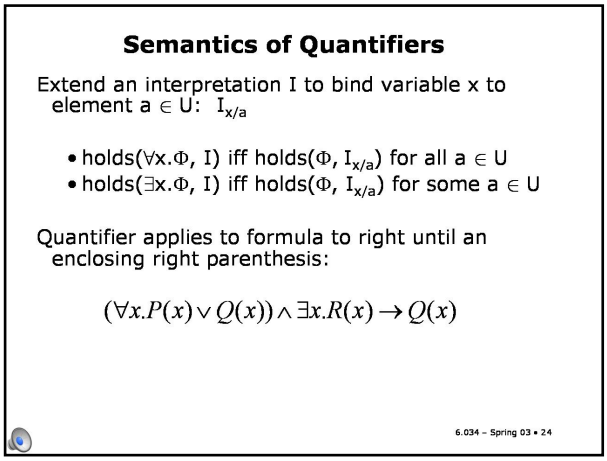




Slide 7.5.22
Similarly, to say that **there exists x such that Phi**, it means that **Phi** has to be true for some **a** in **U**. That is to say, there has to be something in the world such that if we plug that in for **x**, then Phi becomes true.

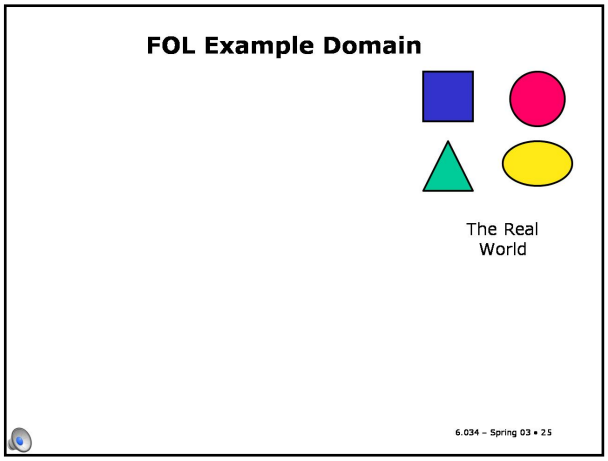
Slide 7.5.23
It's hard to understand the precedence of these operators using the usual rules. A quantifier is understood to apply to everything to its right in the formula, stopping only when it reaches an enclosing close parenthesis.

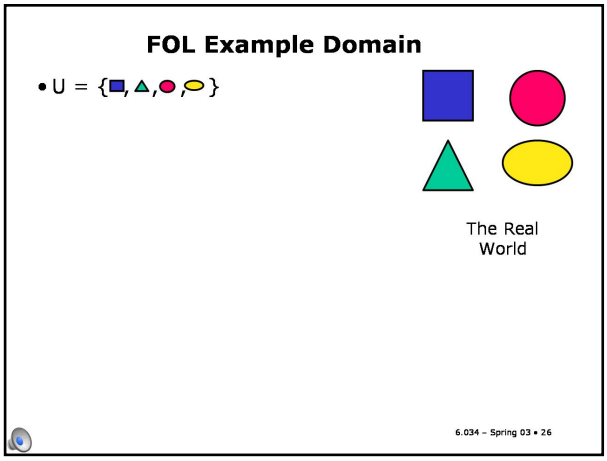




Slide 7.5.24
So in this example sentence, the **for all x** applies until the close paren after the first **Q(x)**; and the **exists x** applies to the end of the sentence.

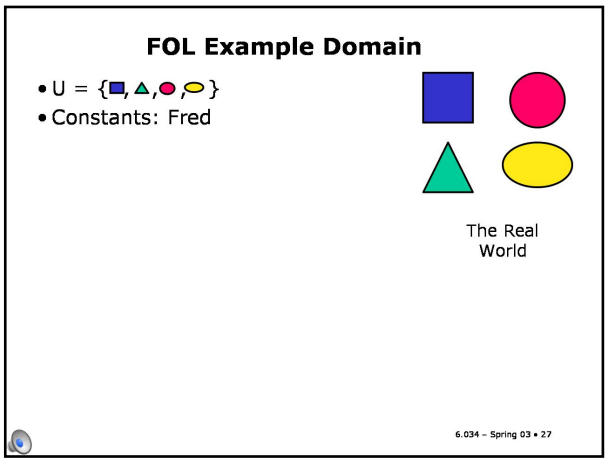
Slide 7.5.25
All right, let's work on an example. Here's a picture of our world.

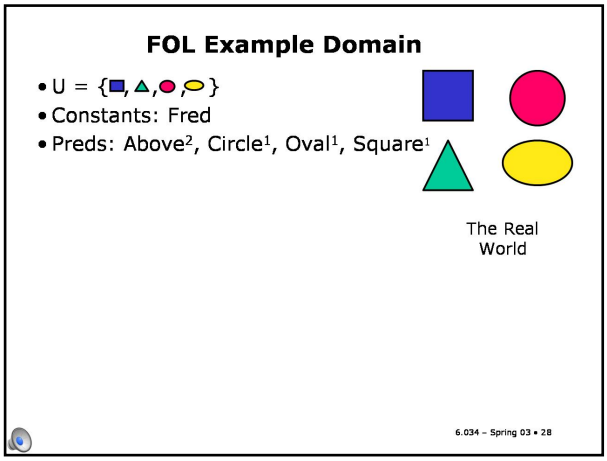




Slide 7.5.26
There are four things in our **U**. Here they are.

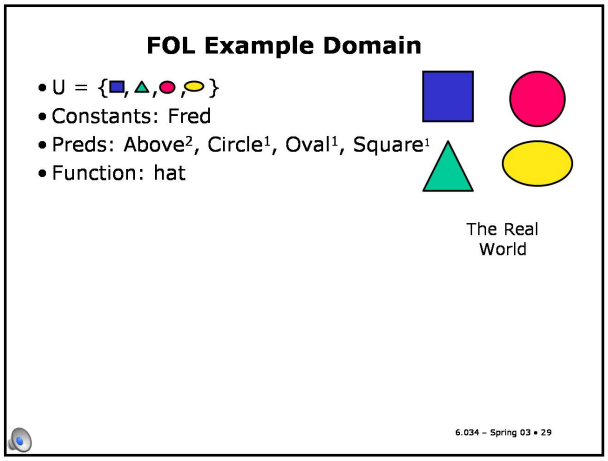
Slide 7.5.27
We have one constant symbol, Fred.

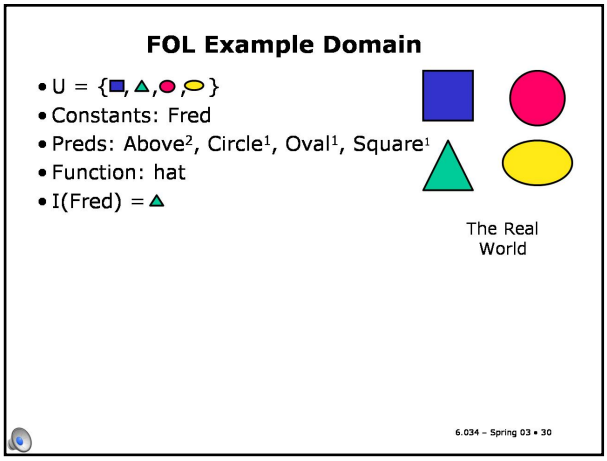




Slide 7.5.28
We have four predicates: **Above**, **Circle**, **Oval**, **Square**. The numbers above them indicate their arity, or the number of arguments they take. Now these particular predicate names suggest a particular interpretation. The fact that I used this word, "circle", makes you guess that probably the interpretation of circle is going to be true for the red object. But of course it needn't be. The fact that those marks on the page are like an English word that we think means something about the shape of an object, that doesn't matter. The syntax is just some words that we write down on our page. But it helps us understand what's going on. It's just like using reasonable variable names in a program that you might write. When you call a variable "the number of times I've been through this loop," that doesn't mean that the computer knows what that means. It's the same thing here.

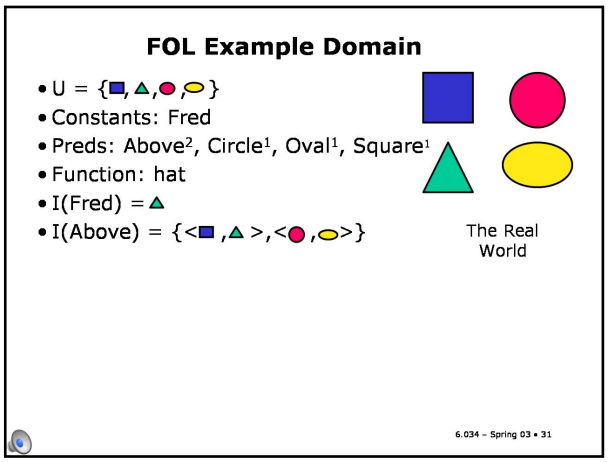
Slide 7.5.29
And we have one function symbol, called **hat**, that takes a single argument.

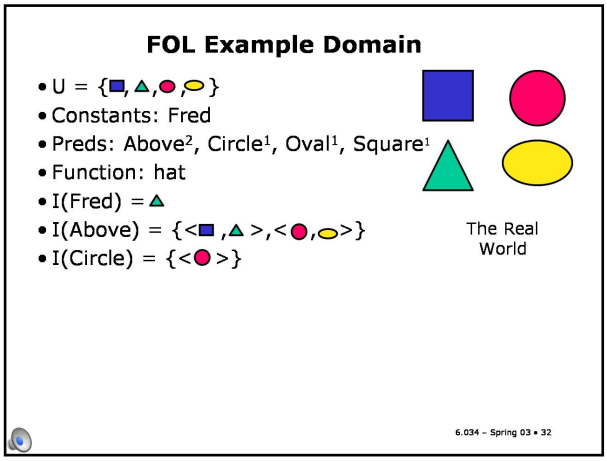




Slide 7.5.30
Now we can talk about a particular interpretation, **I**. We'll define **I** so that **I(Fred)** is the triangle.

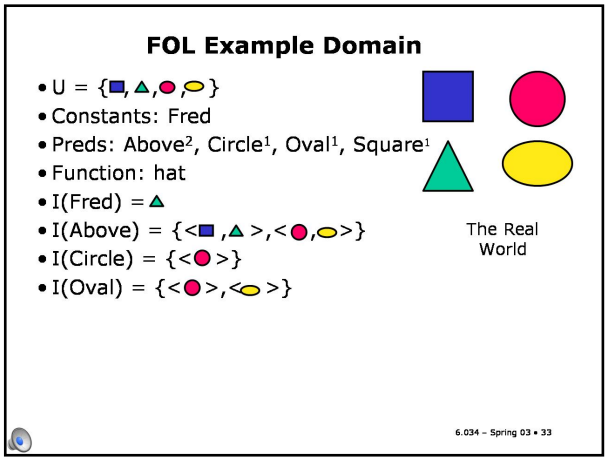
Slide 7.5.31
Now, what kind of a thing is **I(Above)**? Well, **Above** is a predicate symbol, and the interpretation of a predicate symbol is a relation, so **I(Above)** is a relation. Here's the particular relation we define it to be; it's a set of pairs, because **Above** has arity 2. It contains every pair of objects for which we want the relation **Above** to be true.

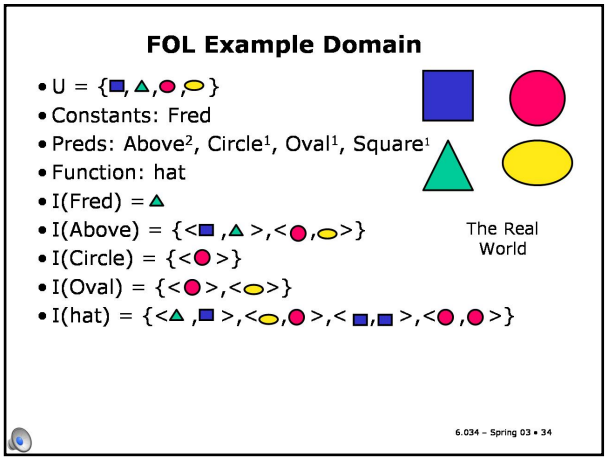




Slide 7.5.32
The interpretation of **Circle** is a unary relation. As you might expect in this world, it's the singleton set, whose element is a one-tuple containing the circle. (Of course, it doesn't have to be!).

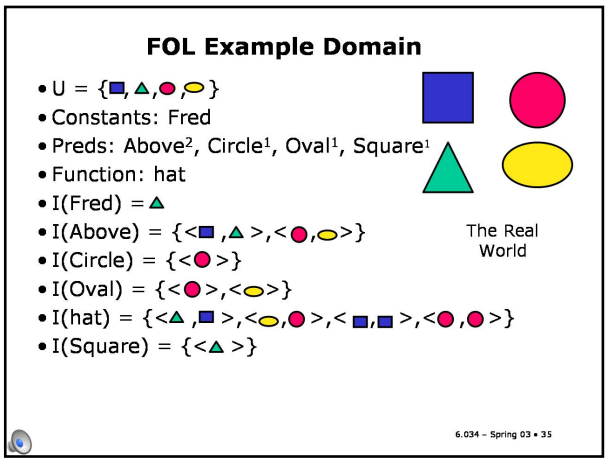
Slide 7.5.33
We'll interpret the predicate **Oval** to be true of both the oval object and the round one (circles are a special case of ovals, after all).

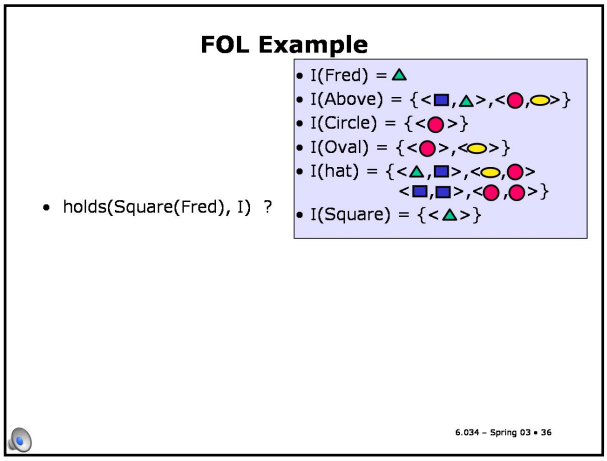




Slide 7.5.34
And we'll say that the hat of the triangle is the square and the hat of the oval is the circle. If we stopped at this point, we would have a function, but it wouldn't be total (it wouldn't have an entry for every possible first argument). So, we'll make it total by saying that the square's hat is the square and the circle's hat is the circle.

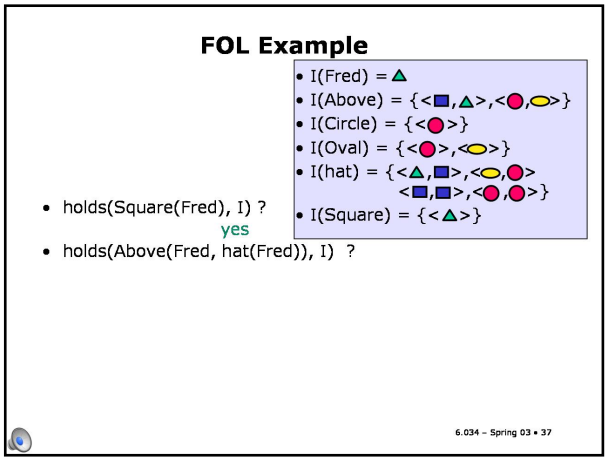
Slide 7.5.35
Finally, just to cause trouble, we'll interpret the predicate **Square** to be true of the triangular object.

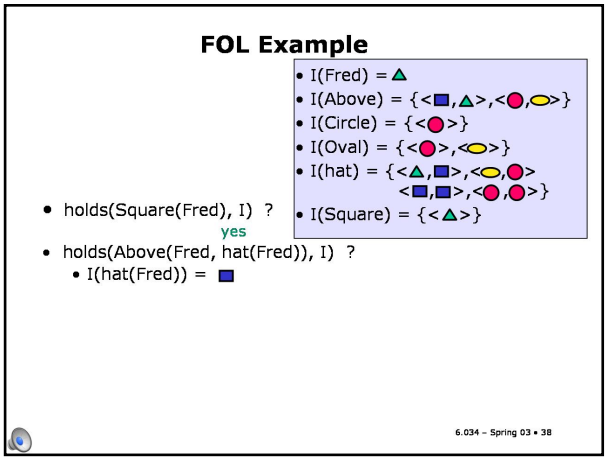




Slide 7.5.36
Now, let's find the truth values of some sentences in this interpretation. What about **Square(Fred)**, is that true in this interpretation? Yes. We look to see that **Fred** denotes the triangle, and then we look for the triangle in the relation denoted by square, and we find it there. So the sentence is true.

Slide 7.5.37
What about this one? Is **Fred** above its hat?

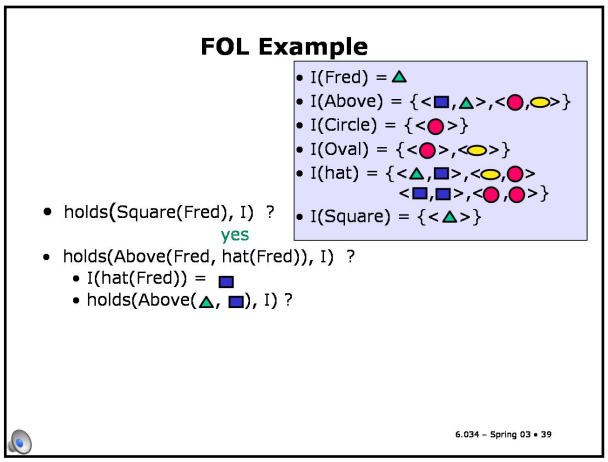


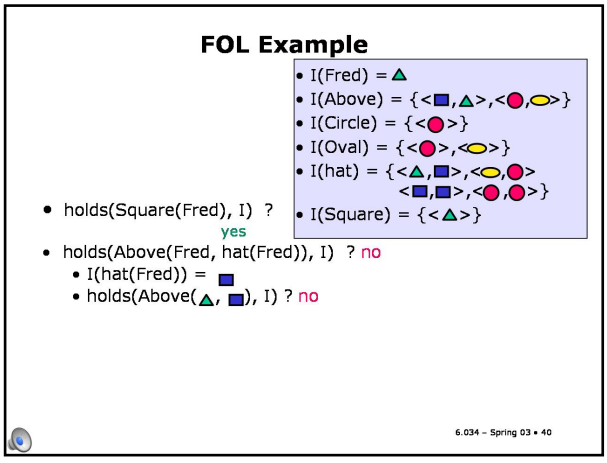


Slide 7.5.38
Let's start by asking the question, what's the denotation of the term, **hat(Fred)**?

It's the square, right? We look up **Fred**, and get the triangle. Then we look in the **hat** function, and, sure enough, there's a pair with triangle first and square second. So **hat(Fred)** is a square.

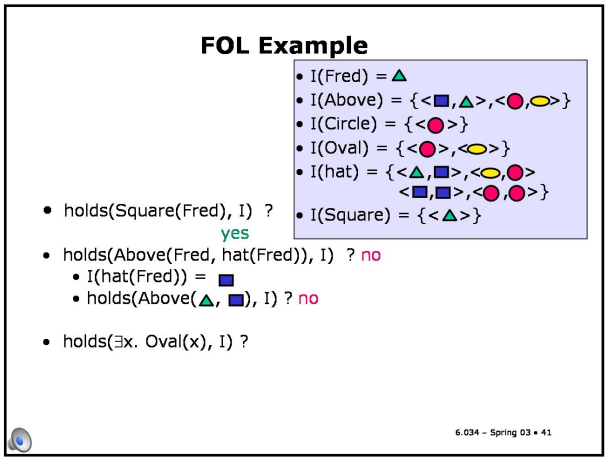
Slide 7.5.39
Now the question is: does the **Above** relation hold of the triangle and the square? We look this pair up in the relation denoted by **Above**, and we can't find it. So the **Above** relation doesn't hold of these objects.

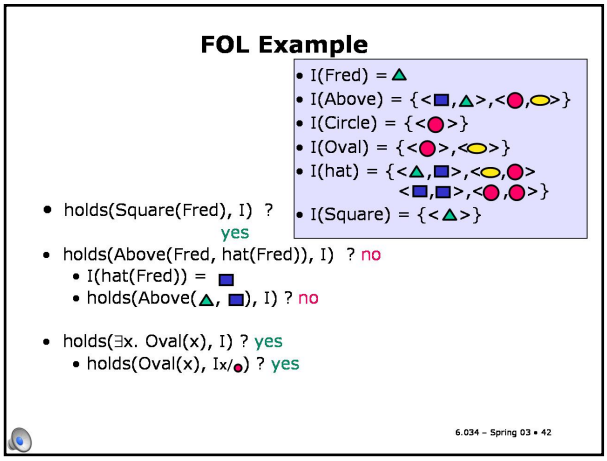




Slide 7.5.40
And our original sentence is false.

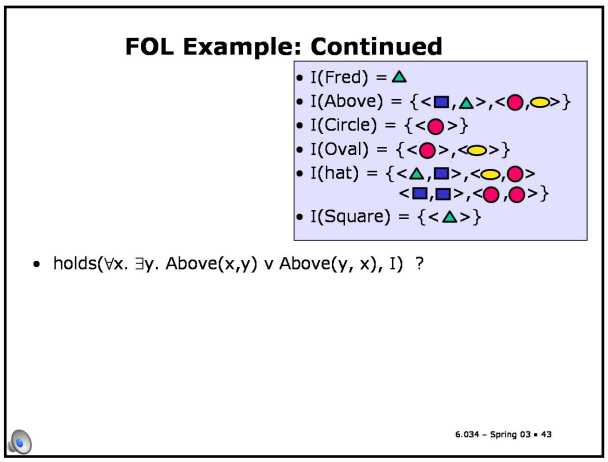
Slide 7.5.41
Okay. What about this sentence: there exists an **x** such that **Oval(x)**. Is there a thing that is an oval? Yes. So how do we show that carefully?

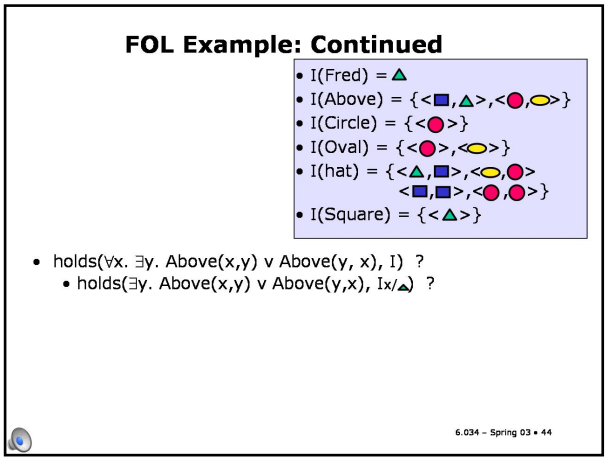




Slide 7.5.42
We say that there's an extension of this interpretation where we take **x** and substitute in for it, the circle. Temporarily, I say that **I(x)** is a circle. And now I ask, in that new interpretation, is it true that **Oval(x)**. So I look up **x** and I get the circle. I look up **Oval** and I get the relation with the circle and the oval, and so the answer's yes.

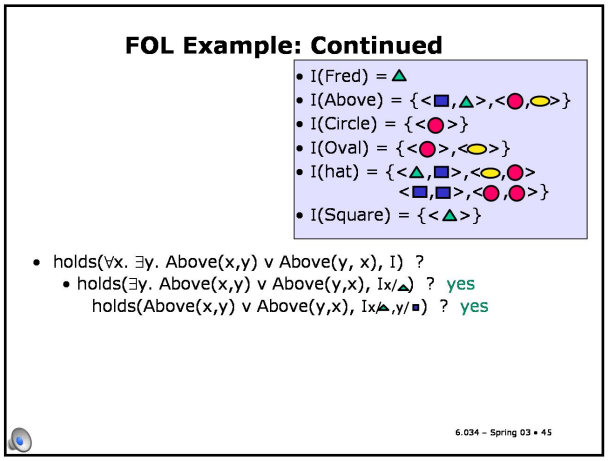
Slide 7.5.43
Here's a more complicated question in the same domain and interpretation. Is the sentence: **For all x there exists a y such that either x is Above y or y is Above x** true in **I**?

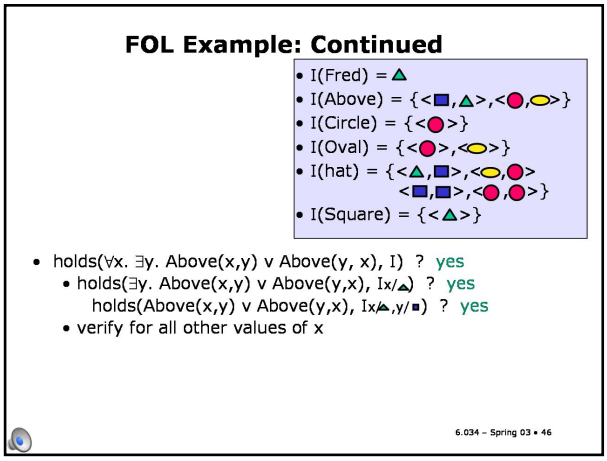




Slide 7.5.44
We can tell whether this is true by going through every possible object in the universe and binding it to the variable **x**, and then seeing whether the rest of the sentence is true. So, for example, we might put in the triangle for **x**, just to start with.

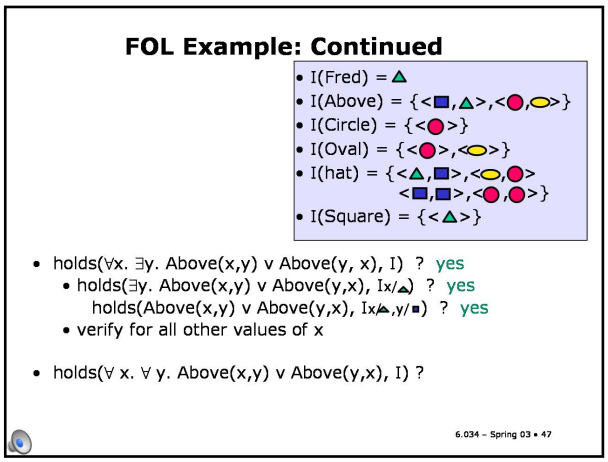
Slide 7.5.45
Now, having made that binding, we have to ask whether the sentence "There exists a y such that either x is above y or y is above x" true in the new interpretation. Existentials are easier than universals; we just have to come up with one y that makes the sentence true. And we can; if we bind **y** to the square, then that makes **Above(y,x)** true, which makes the disjunction true. So, we've proved this existential statement is true.

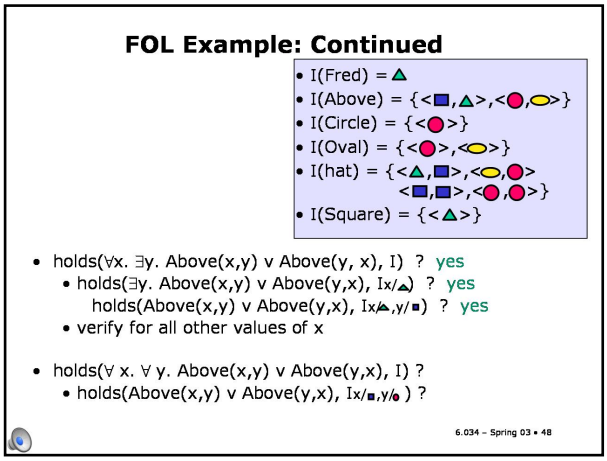




Slide 7.5.46
If we can do that for every other binding of **x**, then the whole universal sentence is true. You can verify that it is, in fact, true, by finding the truth value of the sentence with the other objects substituted in for **x**.

Slide 7.5.47
Okay. Here's our last example in this domain. What about the sentence: "for all x, for all y, x is above y or y is above x"? Is it true in interpretation **I**?

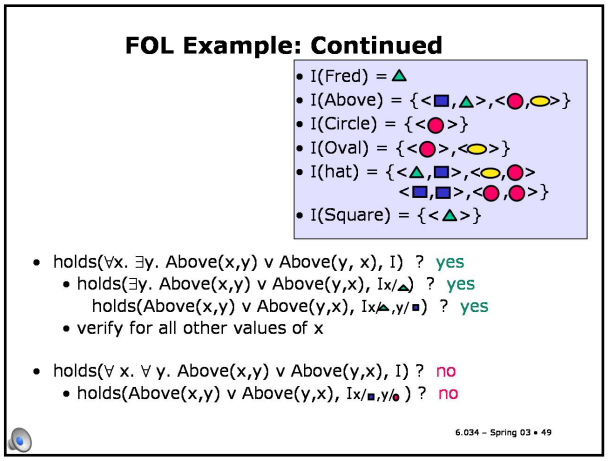




Slide 7.5.48
If it's going to be true, then it has to be true for every possible instantiation of **x** and y to elements of **U**. So, what, in particular, about the case when **x** is the square and **y** is the circle?

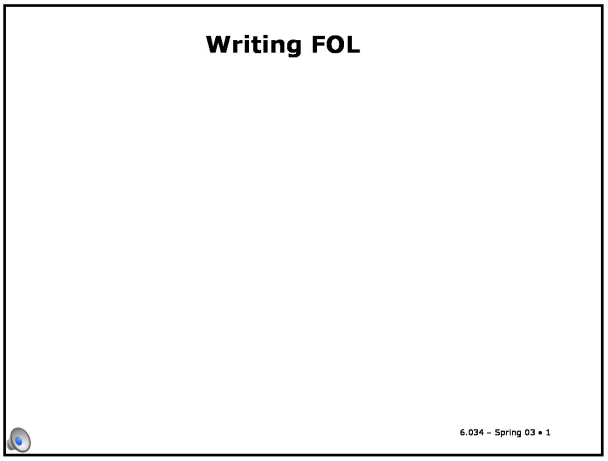
Slide 7.5.49
We can't find either the pair (square, circle), or the pair (circle, square) in the above relation, so this statement isn't true.

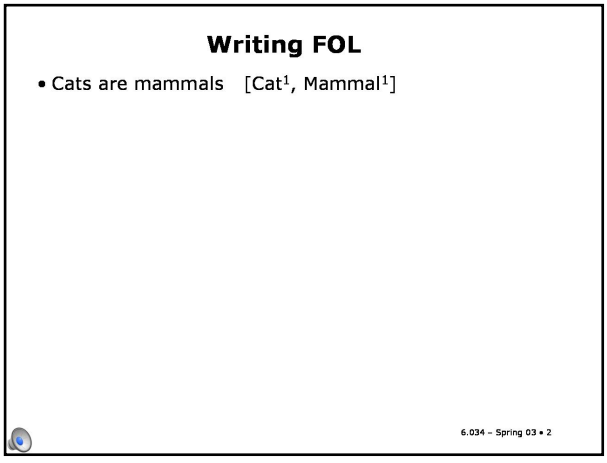
And, therefore, neither is the universally quantified statement.



6.034 Notes: Section 7.6

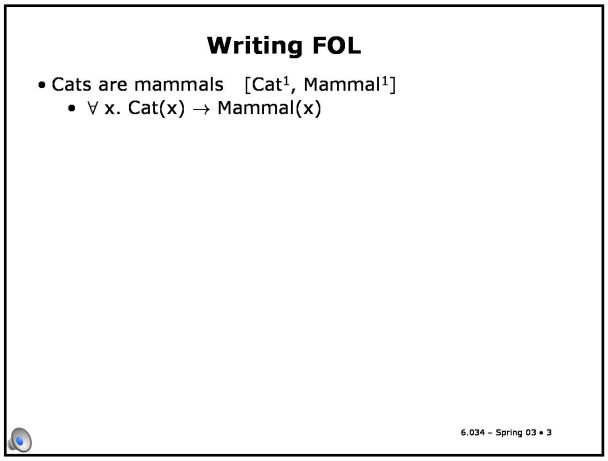
Slide 7.6.1
Now we're going to see how first-order logic can be used to formalize a variety of real-world concepts and situations. In this batch of problems, you should try to think of the answer before you go on to see it.

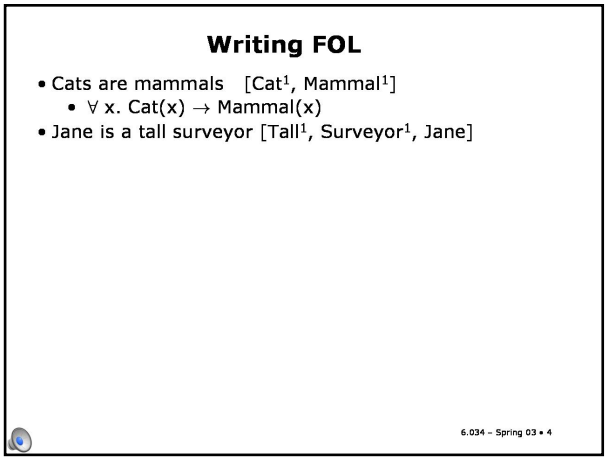




Slide 7.6.2
How would you use first-order logic to say "Cats are mammals"? (You can use a unary predicate **cat** and another unary predicate **mammal**).

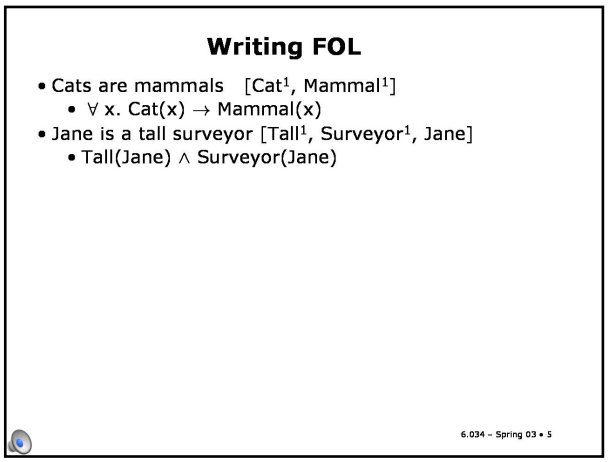
Slide 7.6.3
For all x, Cat(x) implies Mammal(x). This is saying that every individual in the cat relation is also in the mammal relation. Or that cats are a subset of mammals.

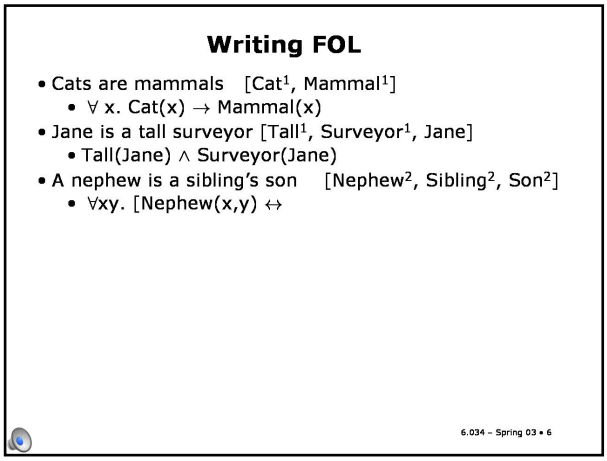




Slide 7.6.4
All right. Let's let **Jane** be a constant, **Tall** and **Surveyor** can be unary predicates. How can we say Jane is a tall surveyor?

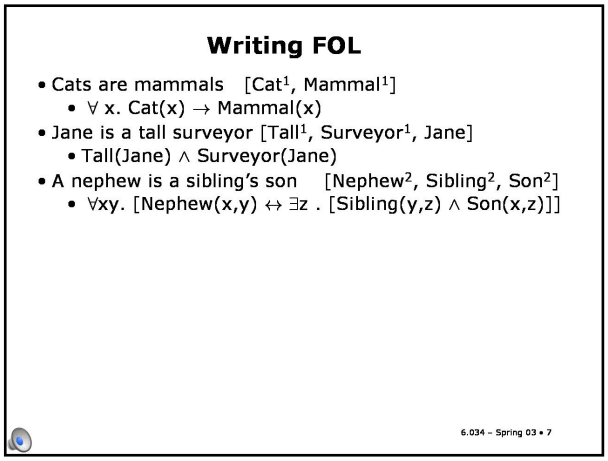
Slide 7.6.5
Surveyor(Jane) and Tall(Jane).

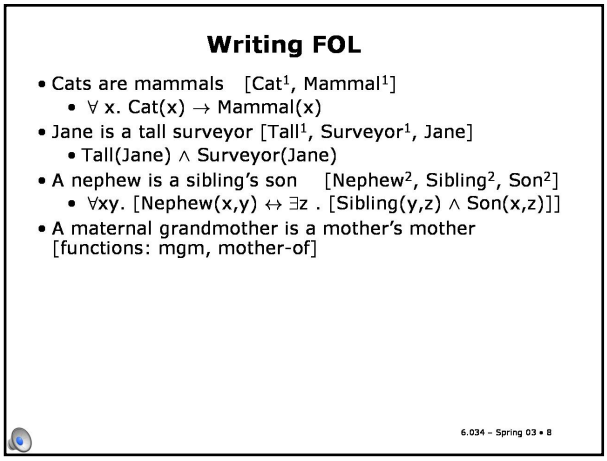




Slide 7.6.6
A nephew is a sibling's son. **Nephew**, **Sibling**, and **Son** are all binary relations. I'll start you off and say **for all x and y, x is the nephew of y if and only if** something. In English, what relationship has to hold between x and y for x to be a nephew of y? There has to be another person z who is a sibling of y and x has to be the son of z.

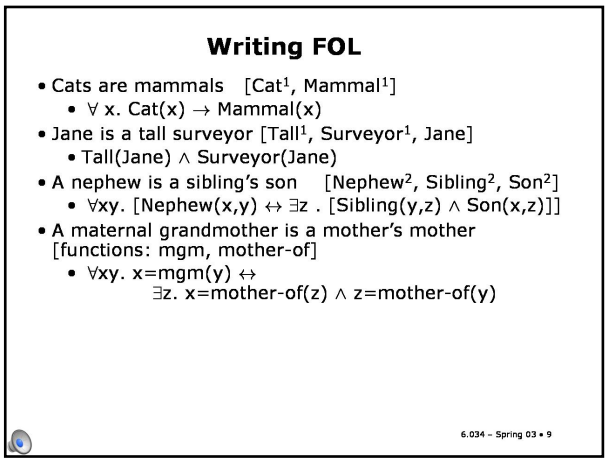
Slide 7.6.7
So, the answer is, "for all x and y, x is the nephew of y if and only if there exists a z such that y is a sibling of z and x is a son of z.

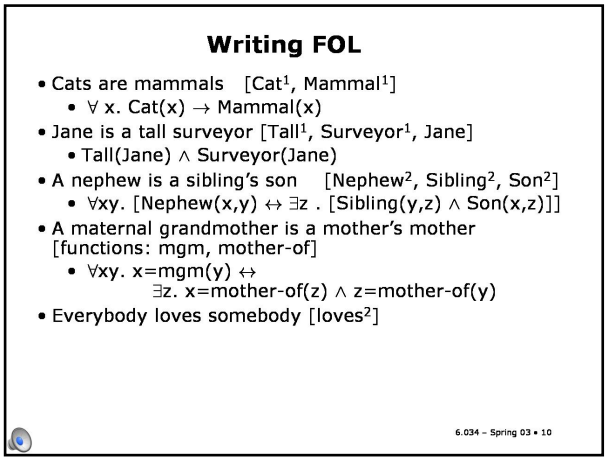




Slide 7.6.8
When you have relationships that are functional, like "mother of", and "maternal grandmother of", then you can write expressions using functions and equality. So, what's the logical way of saying that someone's maternal grandmother is their mother's mother? Use **mgm**, standing for maternal grandmother, and **mother-of**, each of which is a function of a single argument.

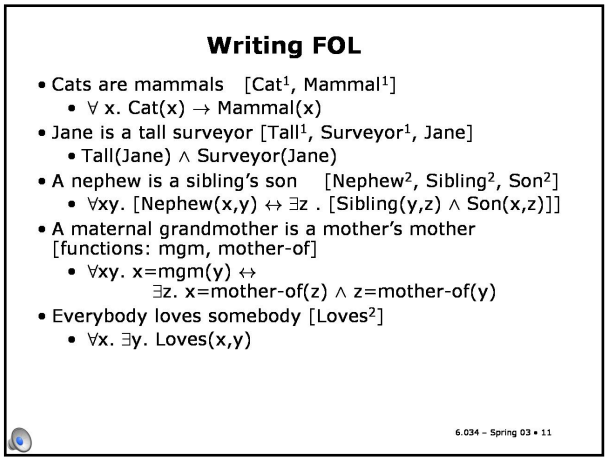
Slide 7.6.9
We can say that, "for all x and y, x is the maternal grandmother of y if and only if there exists a z such that x is the mother of z, and z is the mother of y".

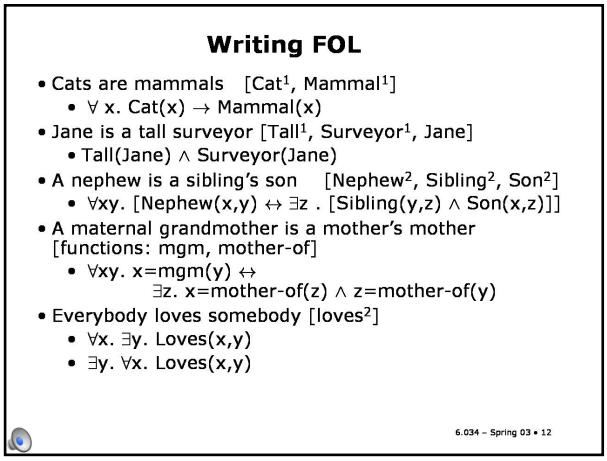




Slide 7.6.10
Using a binary predicate **Loves(x,y)**, how can you say that everybody loves somebody?

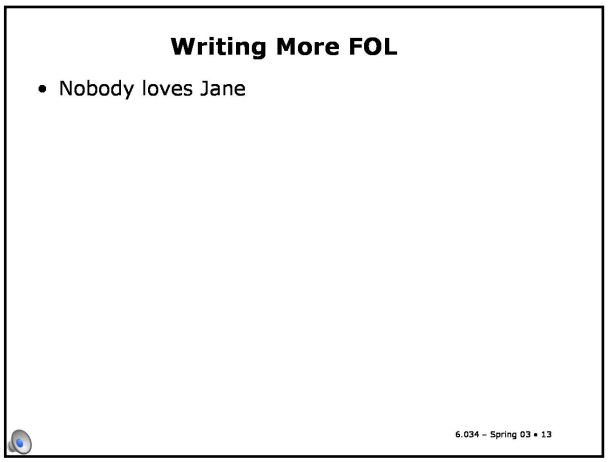
Slide 7.6.11
This one's fun, because there are really two answers. The usual answer is **for all x, there exists a y such that Loves(x,y)**. So, for each person, there is someone that they love. The loved one can be different for each lover. The other interpretation is that there is a particular person that everybody loves. How would we say that?

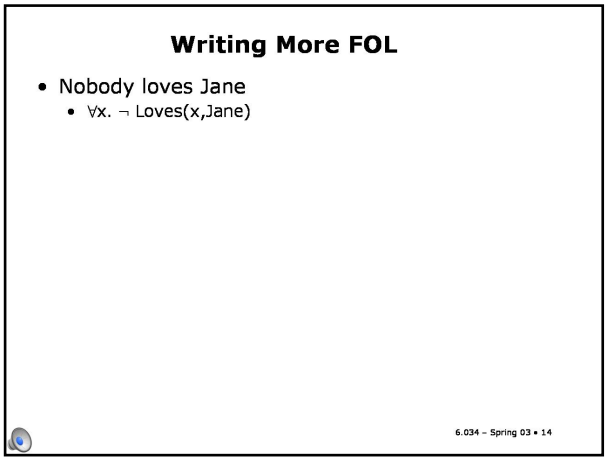




Slide 7.6.12
There exists a y such that for all x, Loves(x,y). So, just by changing the order of the quantifiers, we get a very different meaning.

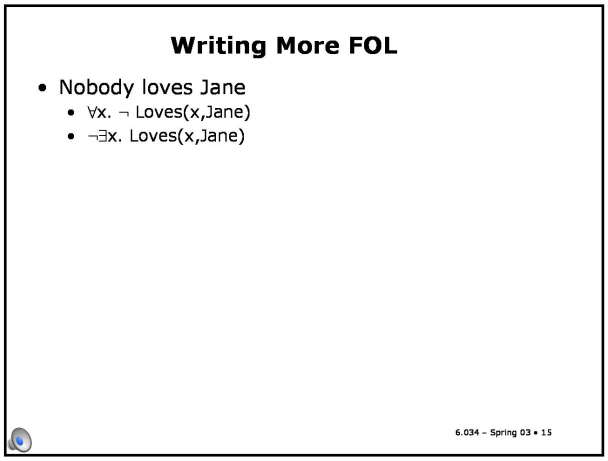
Slide 7.6.13
Let's say nobody loves Jane. Poor Jane. How can we say that?

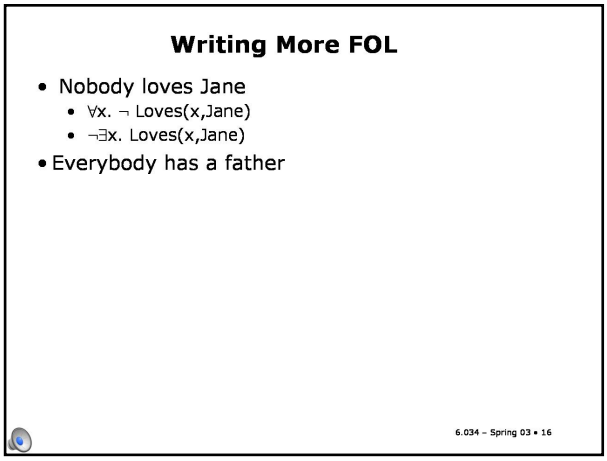




Slide 7.6.14
For all x, not Loves(x, Jane). So, for everybody, every single person, that person doesn't love Jane.

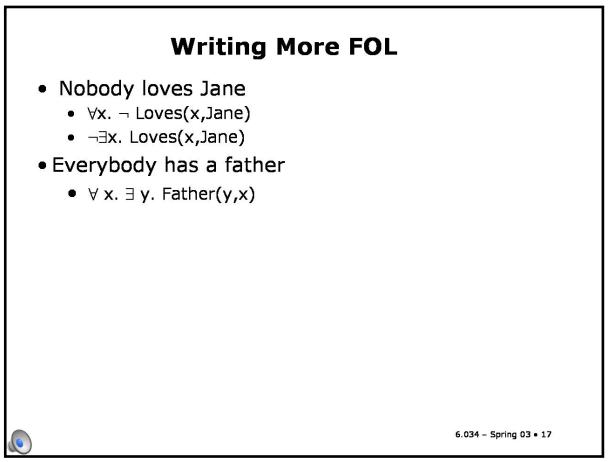
Slide 7.6.15
An equivalent thing to write is there does not exist an **x** such that **Loves(x, Jane)**. This is a general transformation, if you have for all x not something, then it's the same as having not there exists an x something. It's like saying I can't find a single x such that x Loves Jane.

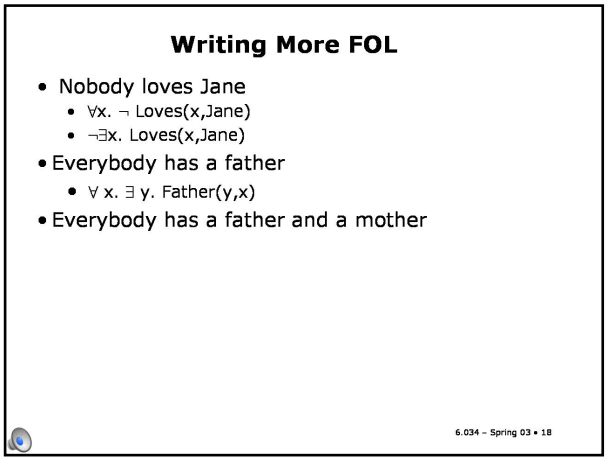




Slide 7.6.16
Everybody has a father.

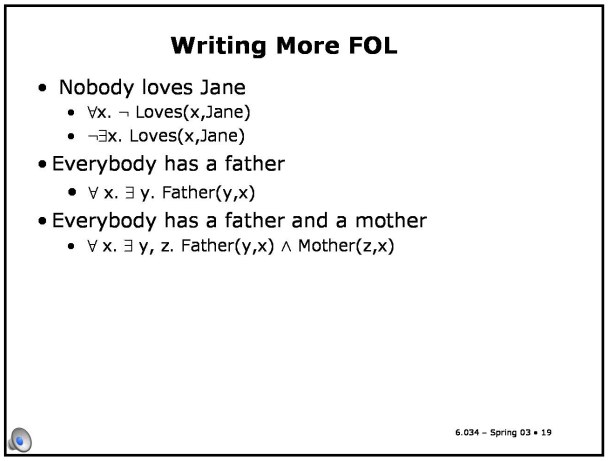
Slide 7.6.17
For all x, exists y such that Father(y,x)

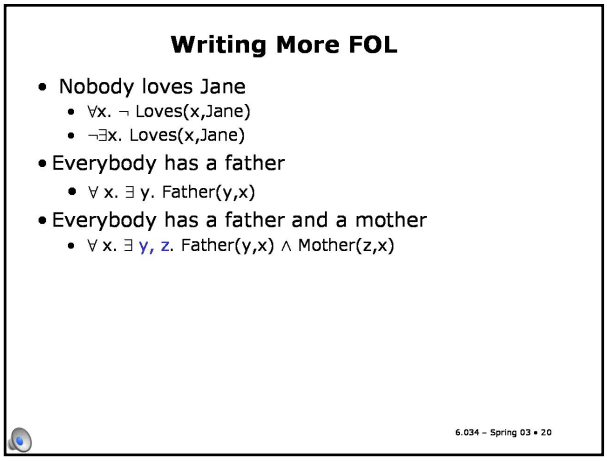




Slide 7.6.18
Everybody has a father and a mother.

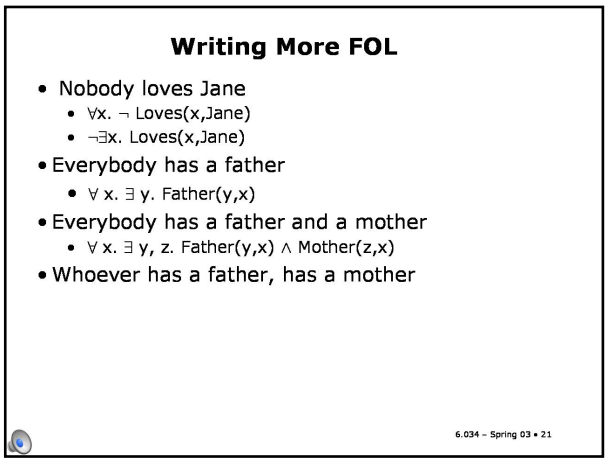
Slide 7.6.19
For all x, exists y, z such that Father(y,x) and Mother(z,x)

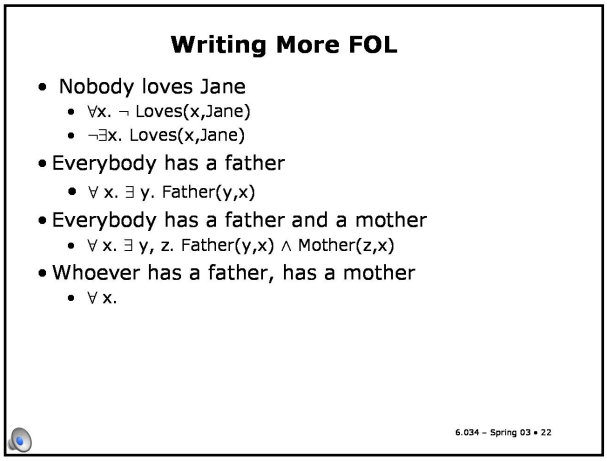




Slide 7.6.20
Now, you might ask whether **y** and **z** are necessarily different. The answer is, no, that's not enforced by the logic. For that matter, they could be the same as **x**. Now, if you use the typical definitions of father and mother, they won't be the same, but that's up to the interpretation.

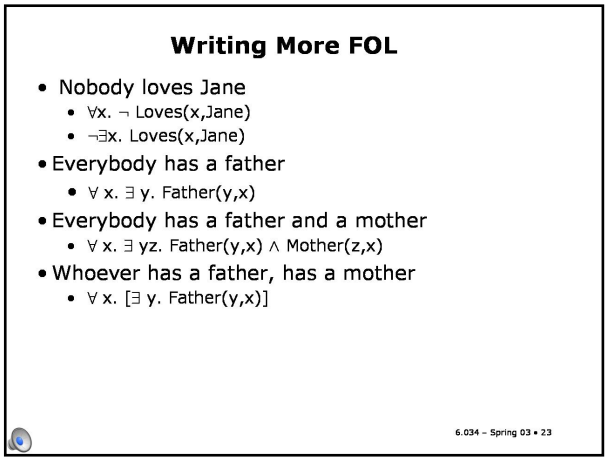
Slide 7.6.21
Whoever has a father has a mother.

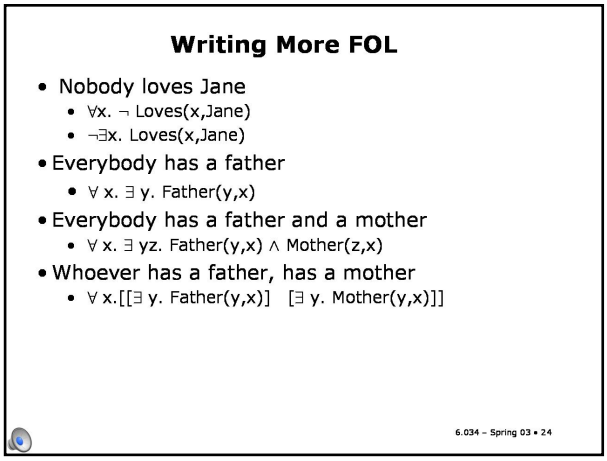




Slide 7.6.22
This is a general statement about objects of the kind, everything that has one property has another property. All right? So we'll talk about everything by starting with **forall x**.

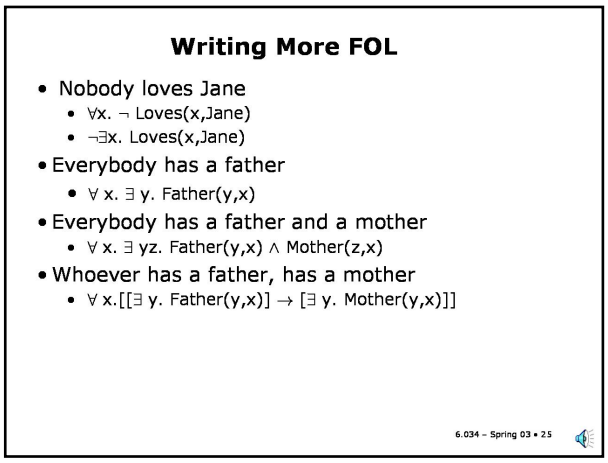
Slide 7.6.23
Now, how do we describe x's that have a father? **Exists y such that Father(y,x)**.

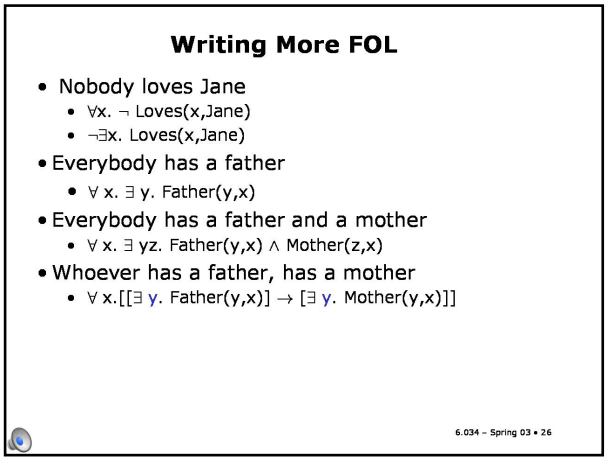




Slide 7.6.24
And we can describe x's that have a mother by **exists y such that Mother (y,x)**.

Slide 7.6.25
Finally, we put these together using implication, just as we did with the "all cats are mammals" example. We want to say objects with a Father are a subset of the set of objects with a Mother (in this case, it will turn out that the sets are equal). So, we end up with "for all x, if there exists a y such that y is the father of x, then there exists a y such that y is the mother of x".

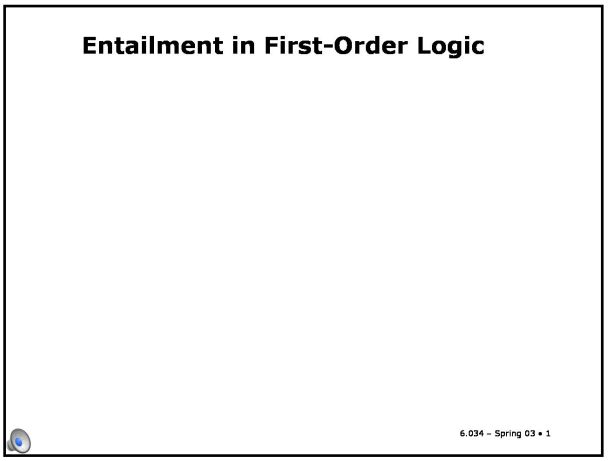


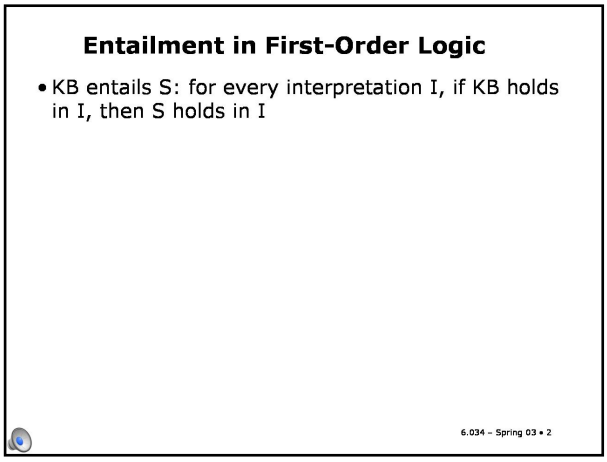


Slide 7.6.26
Note that the two variables named y have separate scopes, and are entirely unrelated to one another. You could rename either or both of them and the semantics of the sentence would remain the same. It's technically legal to have nested quantifiers over the same variable, and there are rules for figuring out what it means, but it's very confusing, so it's just better not to do it.

6.034 Notes: Section 7.7

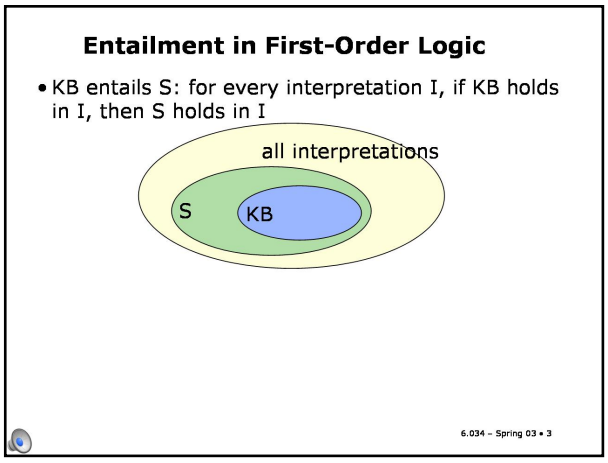
Slide 7.7.1
Now that we understand something about first-order logic as a language, we'll talk about how we can use it to do things. As in propositional logic, the thing that we'll most often want to do with logical statements is to figure out what conclusions we can draw from a set of assumptions. In propositional logic, we had the notion of entailment: a **KB** entails a sentence if and only if the sentence is true in every interpretation that makes **KB** true.

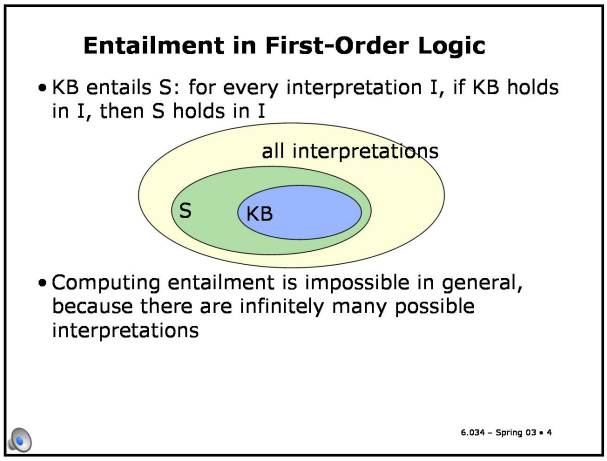




Slide 7.7.2
In first-order logic, the notion of entailment is the same. A knowledge base entails a sentence if and only if the sentence holds in every interpretation in which the knowledge base holds.

Slide 7.7.3
It's important that entailment is a relationship between a set of sentences, **KB**, and another sentence, **S**. It doesn't directly involve a particular intended interpretation that we might have in mind. It has to do with the subsets of all possible interpretations in which **KB** and **S** hold; entailment requires that the set of interpretations in which **KB** holds be a subset of those in which **S** holds. This is sort of a hard thing to understand at first, since the number (and potential weirdness) of all possible interpretations in first-order logic is just huge.



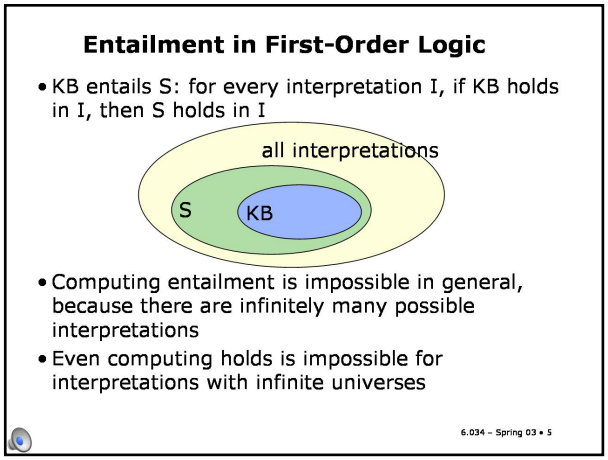


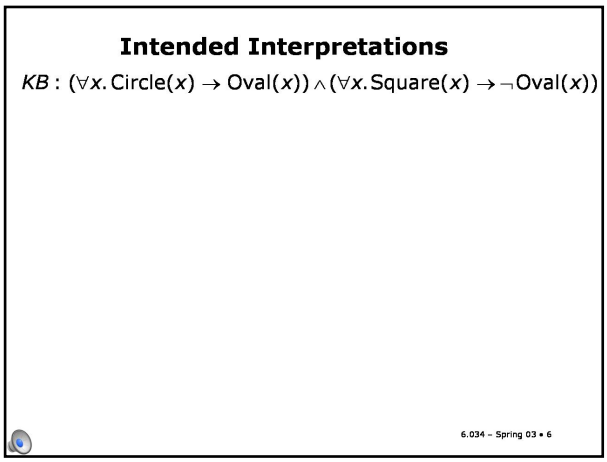
Slide 7.7.4
In propositional logic, we were able to think about computing entailment by doing a brute-force enumeration of all intepretations, then, in each interpretation, checking to see whether the sentence and/or the knowledge base were true in that interpretation.

This will be impossible in first-order logic for two reasons.

First, it's completely out of the question to enumerate all possible interpretations. How many universes are there? More than *I* can count...

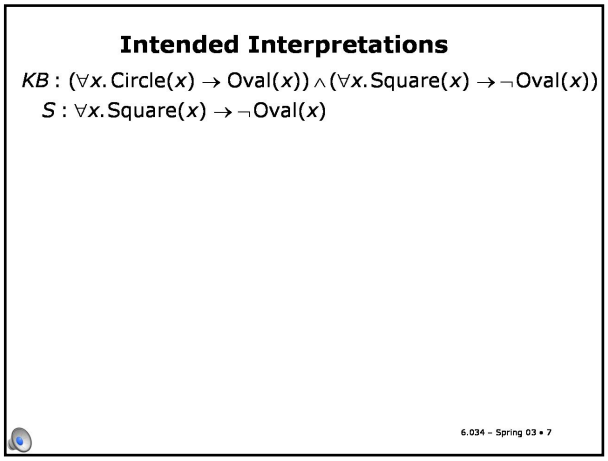
Slide 7.7.5
Second, even for a single interpretation, it's not necessarily possible to compute whether a sentence holds in that interpretation. Why? Because if it has a universal or existential quantifier, it will require testing whether a sentence holds for every substitution of an element in **U** for the quantified variable. And if the universe is infinite, you just can't do that.

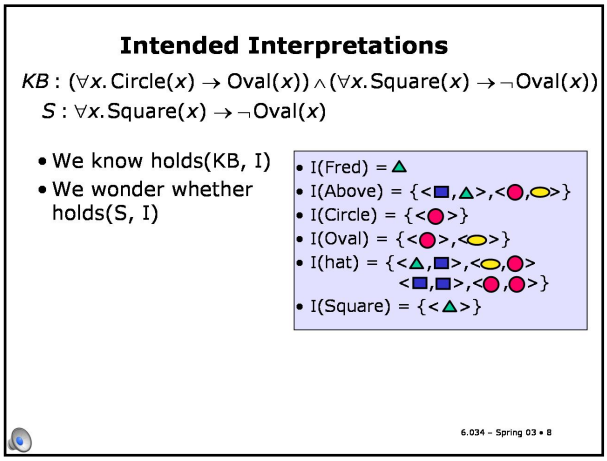




Slide 7.7.6
Let's look at a particular situation in which we might want to do logical inference. Consider our shapes example from before. Let's say that we know, as our knowledge base, that all circles are ovals, and that no squares are ovals. We can write this as **for all x, Circle(x) implies Oval(x)**. And **for all x, Square(x) implies not Oval(x)**.

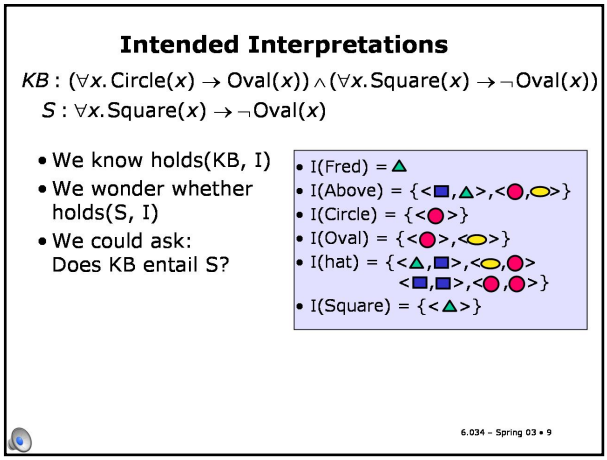
Slide 7.7.7
Now, let's say we're wondering whether it's also true that no squares are circles. We'll call that sentence **S**, and write it **for all x, Square(x) implies not Circle(x)**.

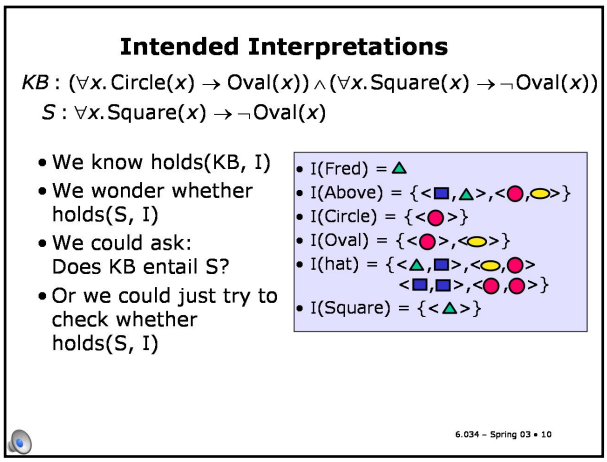




Slide 7.7.8
We know **KB** holds in interpretation **I**, and we wonder whether **S** holds in **I**.

Slide 7.7.9
We could answer this by asking the question: Does **KB** entail **S**? Does our desired conclusion follow from our assumptions.



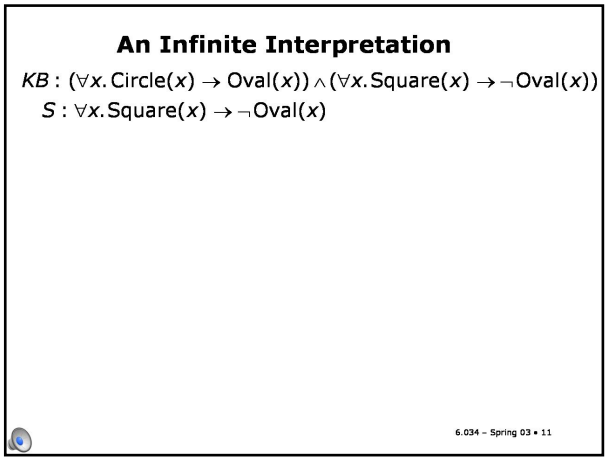


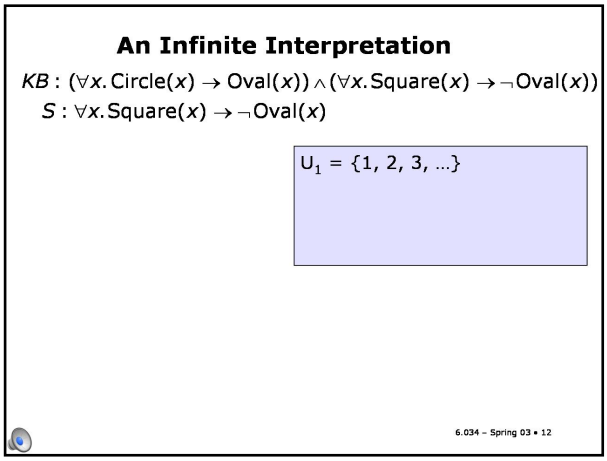
Slide 7.7.10
You might say that entailment is too big a hammer. I don't actually care whether **S** is true in all possible interpretations that satisfy **KB**. Why? because I have a particular interpretation in mind (namely, our little world of geometric shapes, embodied in interpretation **I**). And I know that **KB** holds in **I**. So what I really want to know is whether **S** holds in **I**.

Unfortunately, the computer does not know what interpretation I have in mind. We want the computer to be able to reach valid conclusions about my intended interpretation without my having to enumerate it (because it may be infinite).

For this particular example of **I**, it's not too hard to check whether **S** holds (because the universe is finite and small). But, as we said before, in general, we won't be able even to test whether a sentence holds in a particular interpetation.

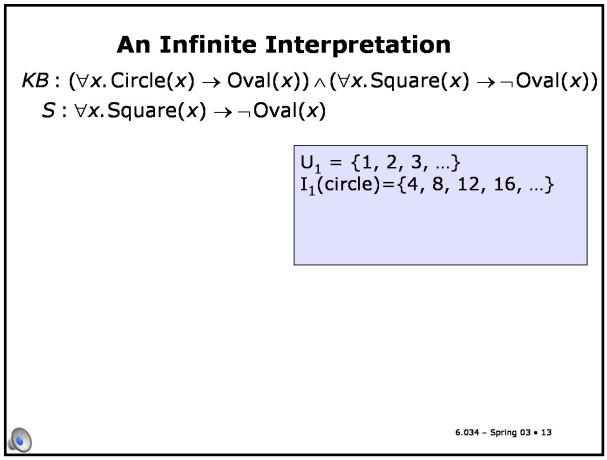
Slide 7.7.11
Let's look at our **KB** for a minute. When we wrote it down, we had a particular interpretation in mind, as evidenced by the names of the propositions. But now, here's another interpretation, $\mathbf{I_1}$

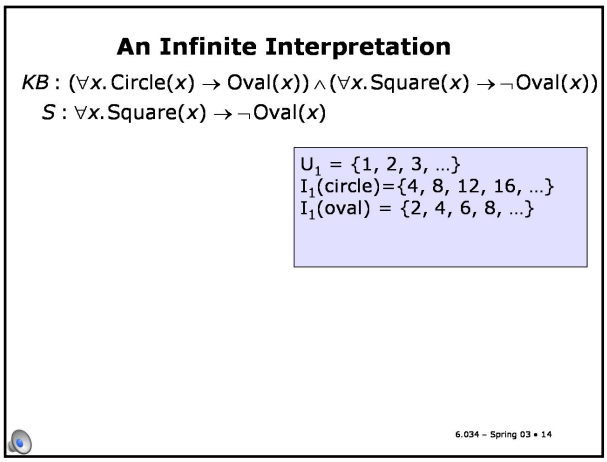




Slide 7.7.12
The universe is the positive integers (numbers 1, 2, etc.). This universe is clearly infinite.

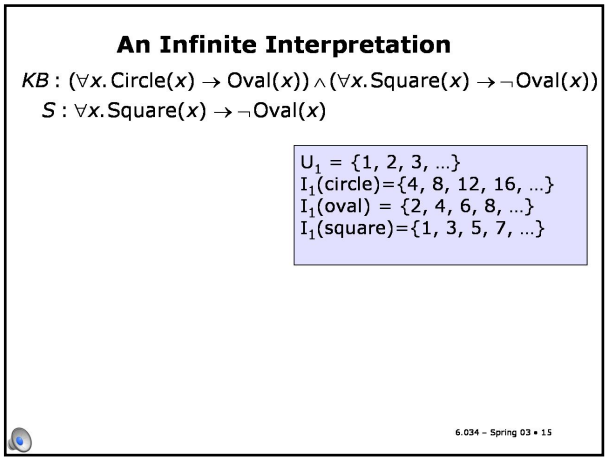
Slide 7.7.13
Let's let the circle relation stand for positive integers evenly divisible by 4. So it's the infinite set {4, 8, 12, 16, ...}.

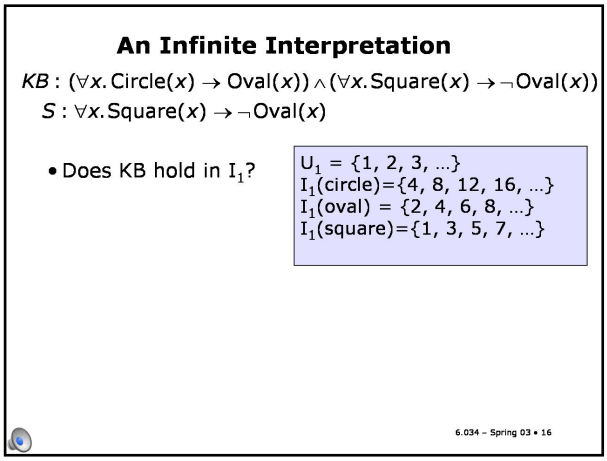




Slide 7.7.14
We'll let **Oval** stand for the even positive integers, {2, 4, 6, 8, ...}.

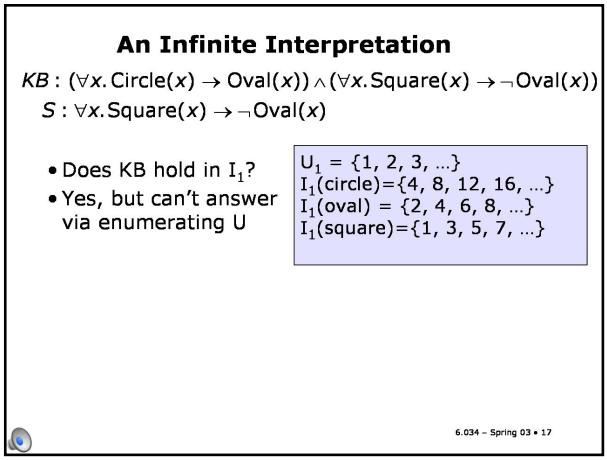
Slide 7.7.15
And we'll let **Square** stand for the odd positive integers, {1, 3, 5, 7, ...}.

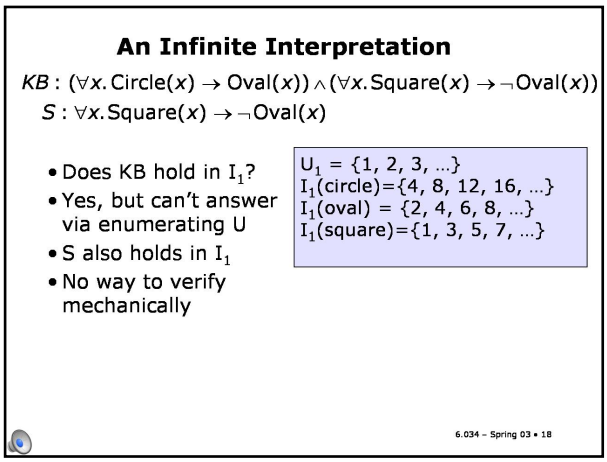




Slide 7.7.16
Now, does **KB** hold in $\mathbf{I_1}$?

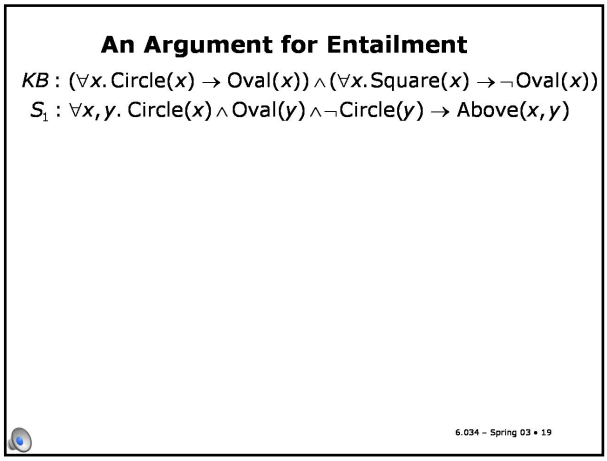
Slide 7.7.17
We can't verify that by enumerating **U** and checking the sentences inside the universal quantifier. However, we all know, due to basic math knowledge, that it does.

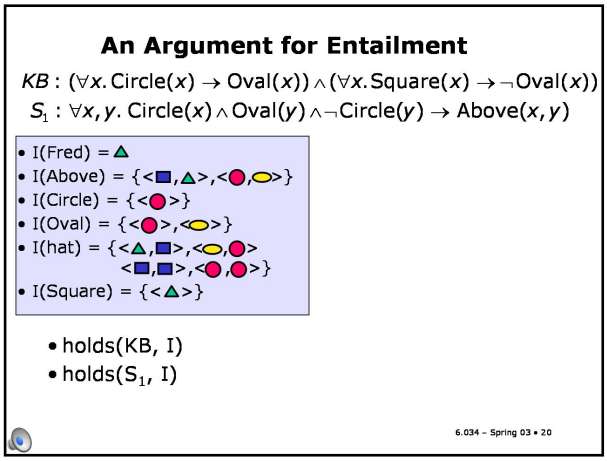




Slide 7.7.18
Similarly, *we* can see that **S** holds in $\mathbf{I_1}$, as well. Unfortunately, we can't rely on our computers to be as smart as we are (yet!). So, if we want a computer to arrive at the conclusion that **S** follows from **KB**, it will have to do it more mechanically.

Slide 7.7.19
Let's think about a different **S**, which we'll call $\mathbf{S_1}$: For every circle and every oval that is not a circle, the circle is above the oval.

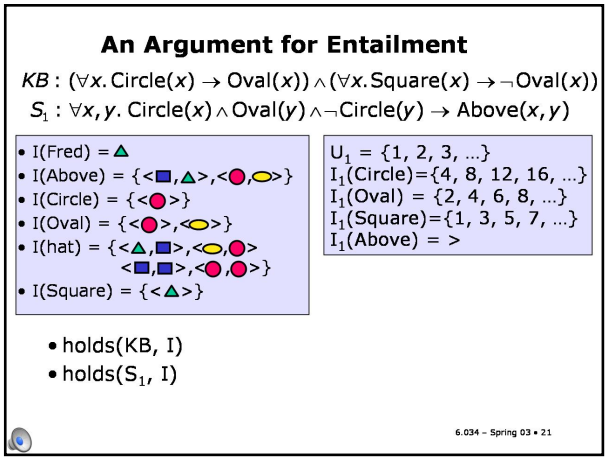


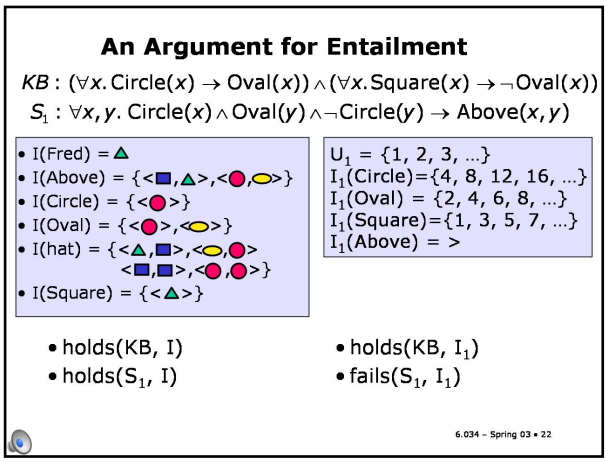


Slide 7.7.20
Back in I, our original geometric interpretation, this sentence holds, right?

But does it "follow from" **KB**? Is it entailed by **KB**?

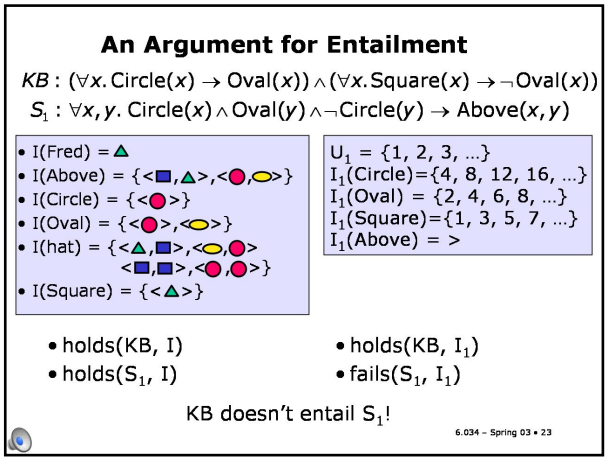
Slide 7.7.21
No. We can see this by going back to interpretation $\mathbf{I_1}$, and letting the interpretation of the "above" relation be greater-than on integers.

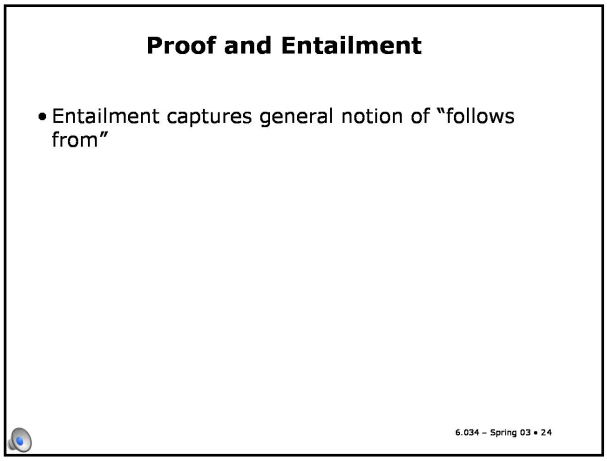




Slide 7.7.22
Then **S** holds in $\mathbf{I_1}$ if all integers divisible by 4 are greater than all integers divisible by 2 but not by 4, which is clearly false.

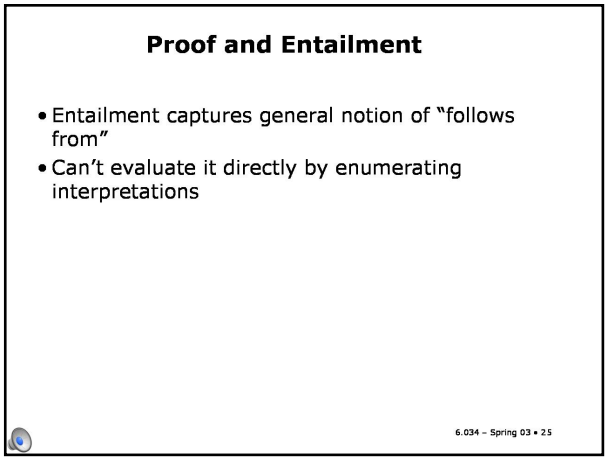
Slide 7.7.23
So, although **KB** and **S** both hold in our original intended interpretation I, **KB** does not entail **S**, because we can find an interpretation in which **KB** holds but **S** does not.

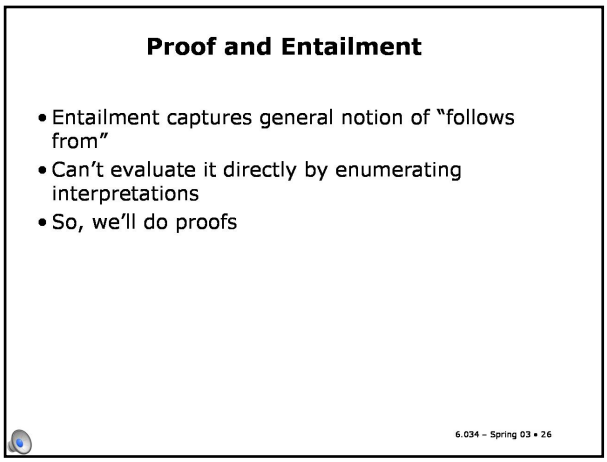




Slide 7.7.24
We can see from this example that entailment captures this general notion of a sentence following from a set of assumptions; of being able to justify the truth of **S** based only on the truth of **KB**.

Slide 7.7.25
So, we like the notion of entailment, but we can't compute it directly.

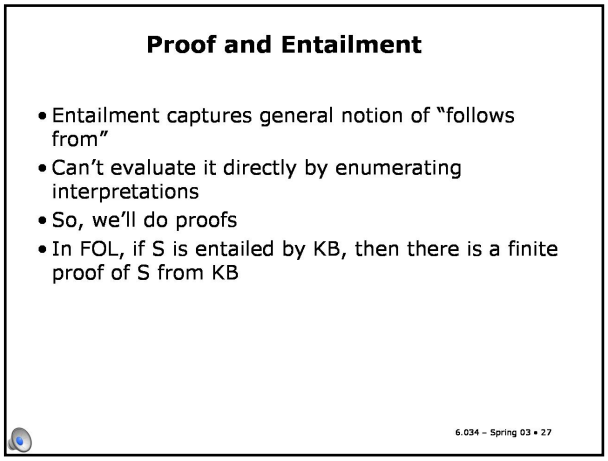


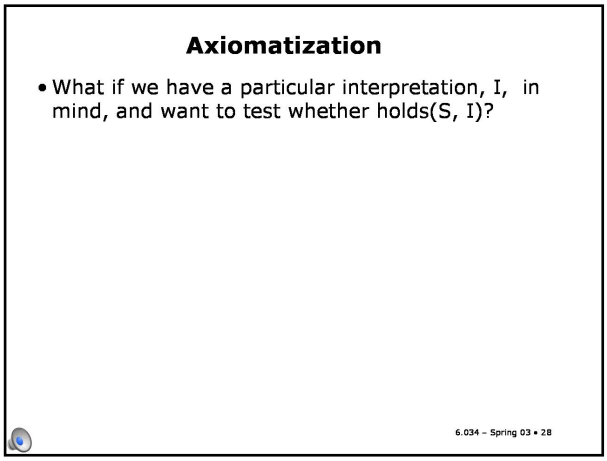


Slide 7.7.26
So what do we do? As we did in propositional logic, we will stay in the domain of syntax, and do proofs to figure out whether **S** is entailed by **KB**.

Slide 7.7.27
There are proof rules that are sound and complete, in the sense that if **S** is entailed by **KB**, there is a finite proof of that. So, it's easier, in general, though not for every particular case, to do a proof of general entailment than to test whether a sentence holds in a given interpretation.

The next few segements of this material will show how to extend the notion of resolution refutation from propositional logic to first-order logic.

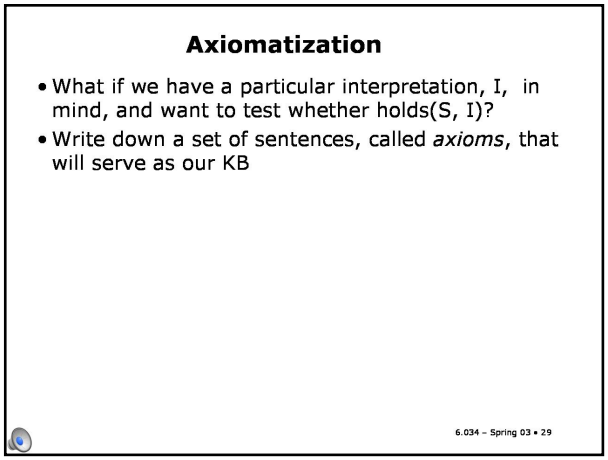


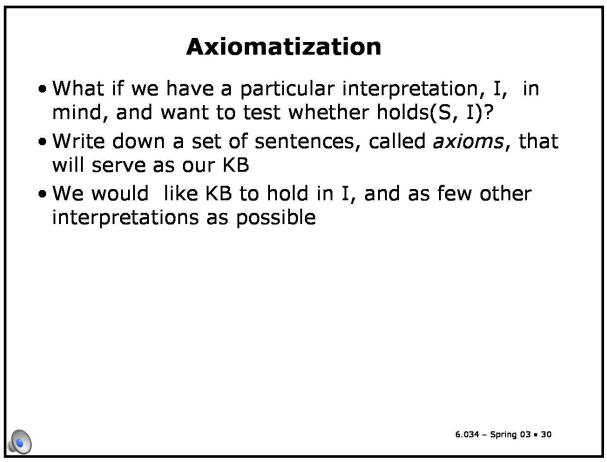


Slide 7.7.28
We just argued that entailment is the right notion when we want to ask the question whether a sentence **S** follows from a **KB**. And that we're going to show entailment via proof.

But what if we have a particular interpretation in mind? We've seen that we can't in general, test whether a sentence holds in that interpretation. How can we use the ability to use proof to show entailment, in order to test whether a sentence holds in an interpretation?

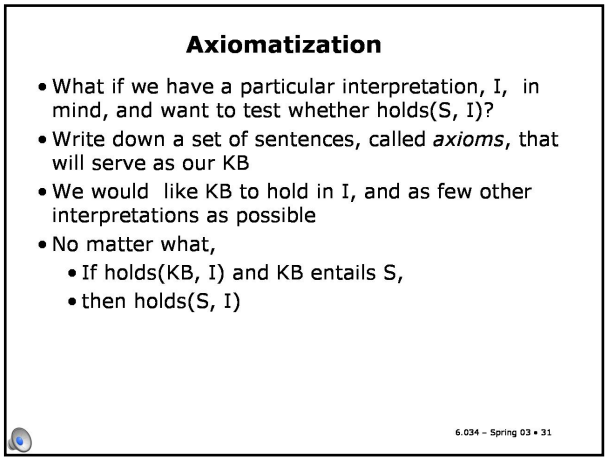
Slide 7.7.29
The answer is that we have to *axiomatize* our domain. That is, we have to write down a set of sentences, or axioms, which will serve as our **KB**.

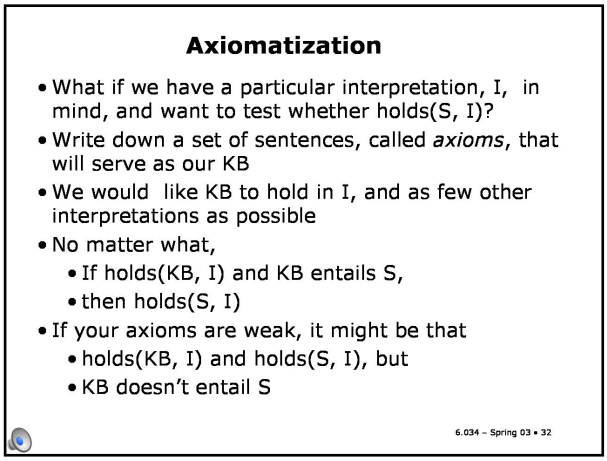




Slide 7.7.30
Ideally the axioms would be so specific that that there was a single interpretation, our intended interpretation, in which they held. In general, though, this will be impossible. You might be able to constrain your axioms to describe domains that contain exactly 4 objects, but you'll never be able to say exactly which 4. You can often give axioms that put stringent enough requirements on the relationships between those objects that all of the interpretations in which the axioms hold are essentially the same as (isomorphic to) your intended interpretation.

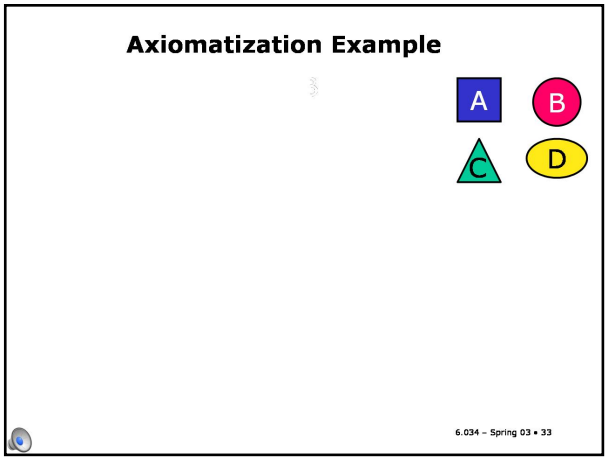
Slide 7.7.31
No matter how constraining your axioms are, you can rely on the fact that if your **KB** holds in your intended interpretation and **KB** entails **S**, then **S** holds in the intended interpretation.

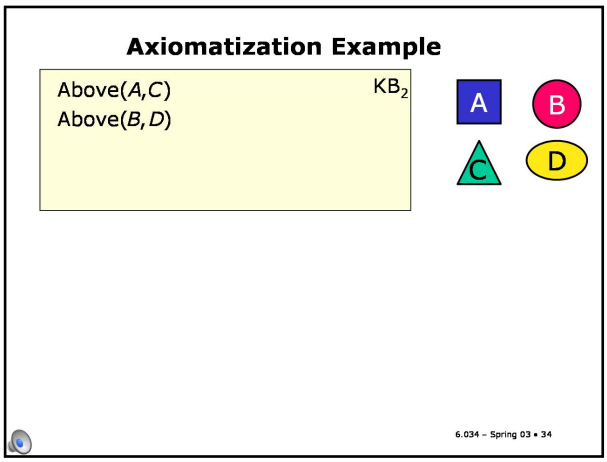




Slide 7.7.32
But that's only half of what we need. There might be some fact, **S**, about your intended interpretation that you would like to be able to derive from your axioms. But, if your axioms are not specific enough, then they might admit some interpretations in which **S** does not hold, and in that case, the axioms will not entail **S**, even though it might hold in the intended interpretation.

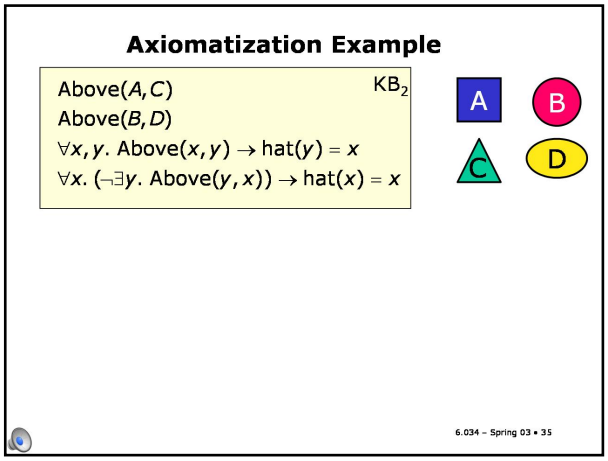
Slide 7.7.33
Let's work through an example of axiomatizing a domain. We'll think about our good old geometric domain, but, to simplify matters a bit, let's assume that we have the constant symbols **A**, **B**, **C**, and **D**. And let our interpretation specify that **A** is the square, **B** is the circle, **C** is the triangle, and **D** is the oval.

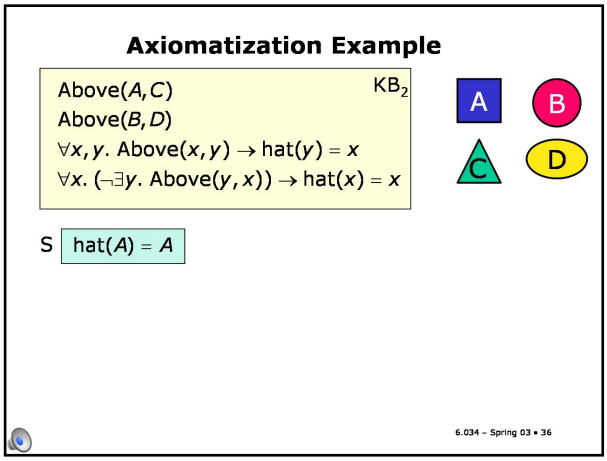




Slide 7.7.34
We propose to axiomatize this domain by specifying the above relation on these constants: **Above(A,C) and Above (B, D)**.

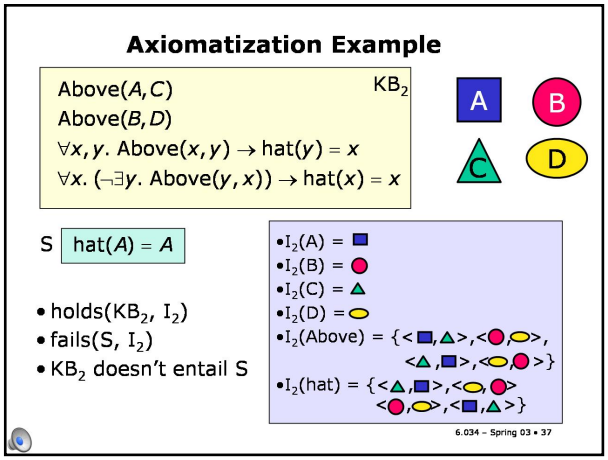
Slide 7.7.35
And we'll give some axioms that say how the hat function can be derived from **Above**: "for all x and y, if x is above y, then hat of y equals x; and for all x, if there is no y such that y is above x, then hat of x equals x".

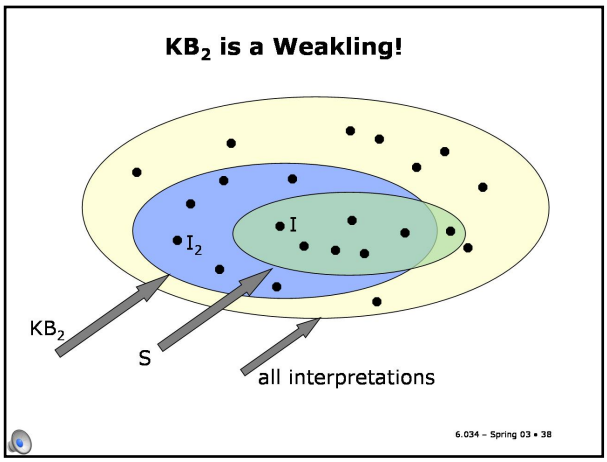




Slide 7.7.36
These four axioms will constitute our **KB**. Now, we're curious to know whether it's okay to conclude that the hat of **A** is **A**. It's true in our intended interpretation, and we'd like it to be a consequence of our axioms.

Slide 7.7.37
So, does our **KB** entail **S**? Unfortunately not. Consider the interpretation $\mathbf{I_2}$. It has two extra pairs in the interpretation of Above. Our axioms definitely hold in this interpretation, but **S** does not. In fact, in this interpretation, the sentence **hat(A) = C** will hold.

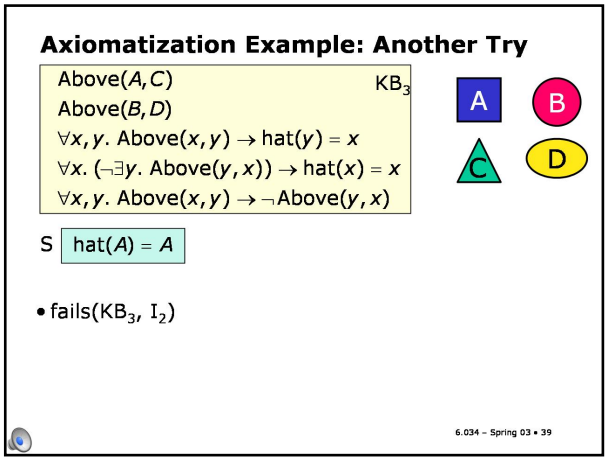


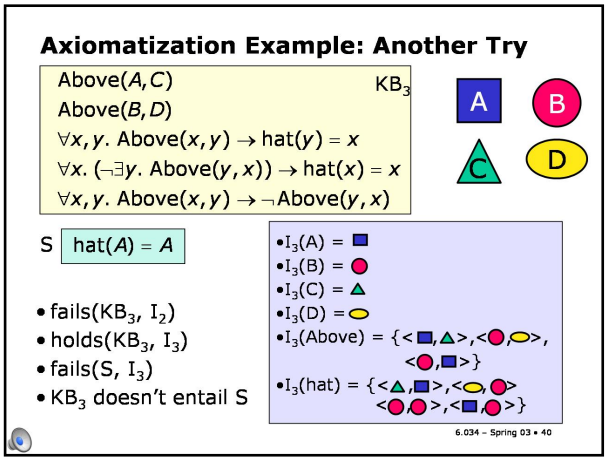


Slide 7.7.38
Just so we can see what's going on, let's go back to our Venn diagram for entailment. In this case, the blue set of interpretations in which the **KB** holds is not a subset of the green set of interpretations in which **S** holds. So, it is possible to have an interpretation, $\mathbf{I_2}$, in which **KB** holds but not **S**. **KB** does not entail **S** (for it to do so, the blue area would have to be a subset of the green), and so we are not licensed to conclude **S** from **KB**.

How can we fix this problem? We need to add more axioms, in order to rule out $\mathbf{I_2}$ as a possible interpretation. (Our goal is to make the blue area smaller, until it becomes a subset of the green area).

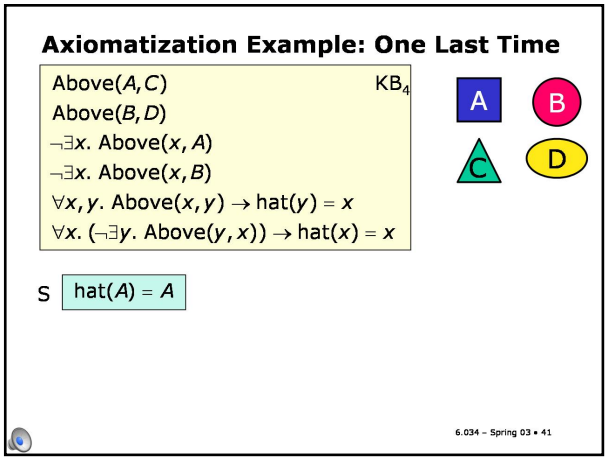
Slide 7.7.39
Here's a reasonable axiom to add: "for all x and y, if x is above y then y is not above x". It says that above is asymmetric. With this axiom added to our **KB**, **KB** no longer holds in $\mathbf{I_2}$, and so our immediate problem is solved.

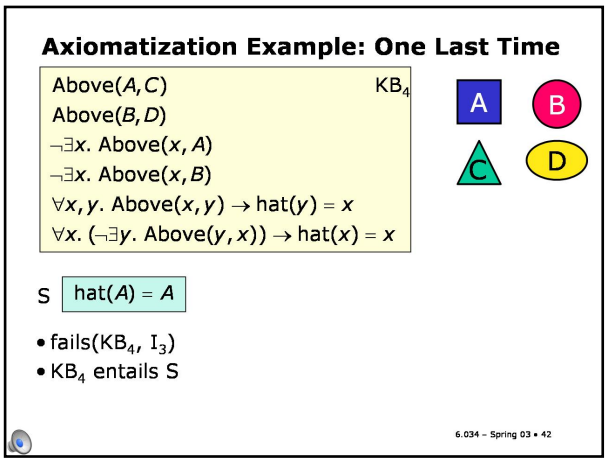




Slide 7.7.40
But we're not out of the woods yet. Now consider interpretation $\mathbf{I_3}$, in which the circle is above the square. **KB** holds in $\mathbf{I_3}$, but **S** does not. So **S** is still not entailed by $\mathbf{I_3}$.

Slide 7.7.41
Clearly, we're missing some important information about our domain. Let's add the following important piece of information to our set of axioms: there is nothing above the square or the circle.





Slide 7.7.42
If we let our new **KB** have these axioms as well, then it fails in $\mathbf{I_3}$, and does, in fact, entail **S**. Whew.

Slide 7.7.43
So, when you are axiomatizing a domain, it's important to be as specific as you can. You need to find a way to say everything that's crucial about your domain. You will never be able to draw false conclusions, but if you are too vague, you may not be able to draw some of the conclusions that you desire.

It turns out, in fact, that there is no way to axiomatize the natural numbers without including some weird unintended interpretations that have multiple copies of the natural numbers.

Still this shouldn't deter us from the enterprise of using logic to formalize reasoning inside computers. We don't have any substantially better alternatives, and, with care, we can make logic serve a useful purpose.