

1 More on Predicate Calc

- Last time, we began looking at the predicate calculus
- As a refresher, statements in the P.C. are made up of 4 elements:
 1. **Constants** like `frodo`, `mount_mckinley`, `-3`, etc. (Note that constants are traditionally written lower-case.)
 2. The special reserved values `TRUE` and `FALSE`
 3. **Variables**, like `X`, `Y`, etc.
 4. **Functions** like `plus/2`, `lcd/2`, etc. Functions can have arbitrary arity, but functions of arity 0 are effectively constants.
 5. **Predicates** like `father/2`, `evil/1`, `equal/2`, `likes/2`, etc. Predicates can also have arbitrary arity, though a predicate of arity 0 is effectively equivalent to either `TRUE` or `FALSE`.
 6. **Logical operations**: $\wedge, \vee, \neg, \rightarrow, \equiv$ with the standard semantics. (Sometimes we write \leftrightarrow instead of \equiv .)
 7. **Quantifiers**: \forall, \exists
- We pretty much know what the semantics of these things are, though we'll go through them a bit.
- Aside: do we all know the distinction between *syntax* and *semantics*?
- A couple of more definitions that we'll need:
 - A **term** is either a variable, a constant, or a function expression. That is, a term effectively represents an object or thing in the world.
 - An **atom** is a predicate instantiated with its appropriate arguments (e.g., if `father/2` is a predicate, then **`father(X,bilbo)`** is an atom.) An atom is the smallest thing with a truth value in P.C., and it's the smallest thing that you can operate on with logical operations and quantifiers. (Note that `TRUE` and `FALSE` are, therefore, also atoms.)
 - A **sentence** is a set of atoms interconnected in the standard way with logical connectives. I won't go through that in detail today – details are in your book. Basically: don't make syntax errors.
 - A **ground expression** has no variables in it at all – everything is a constant.
 - A variable can be either **quantified** or **free** – in the P.C., all variables must be quantified.
- One way to think of predicates is as *relations* – they describe which things in the world (terms) are related and how.
- You can think of an n -ary predicate as, effectively, being a bit for every tuple of n items from the universe of constants. I.e., if your universe is `frodo`, `gandalf`, and `sauron`, then the predicate `evil/1` can be interpreted as:

frodo	gandalf	sauron
0	0	1

while the predicate `friends/2` can be interpreted as:

	frodo	gandalf	sauron
frodo	1	1	0
gandalf	1	1	0
sauron	0	0	0

(Sauron is such a rotten guy that he doesn't like *anybody* – not even himself.)

- We're working with **first order** predicate calculus here (sometimes called first order logic, or First Order Predicate Logic==FOPL). First order means that you can quantify over *variables* but not *predicates*. *Second order* P.C. would allow you to quantify over predicates.
- There are a couple of things that we want to be able to do with a language like FOPL: *represent* things – write down expressions about the world, and *reason* with it.
- Propositional logic gave us, basically, the ability to describe *facts* about the world. FOPL, lets us describe *facts*, *objects*, and *relationships* between those objects. Much richer language.
- In fact, we can use a few statements in FOPL to express an infinite world (see overhead).
- These are the *Peano axioms* which define all of natural number arithmetic. We have just encoded, effectively, all of natural arithmetic and number theory in 5 axioms.
- Clearly, proving things in FOPL is no easier than proving things in number theory.
- So let's look at how we might actually go about proving things given a bunch of statements (sentences) in FOPL.
- Much like our state space searches previously, we'll start with an initial setting and extend it by applying "successor" functions.
- In this case, our "successors" are called **inference rules** (also, **production rules** or **rewrite rules**).
- You've already seen an inference rule before – DeMorgan's law: $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
- This says that any time we see something of the form " $\neg(P \wedge Q)$ ", we can rewrite it to the other form.
- You can imagine that if you have a pile of such rewrite rules, you can use them to rewrite all your axioms in sequence until you find one that matches your goal.
- Example: probably the oldest inference rule that you're all familiar with is **modus ponens** (Latin, lit. "mood that affirms"):

$$\begin{array}{c}
 P \\
 P \rightarrow Q \\
 \hline
 Q
 \end{array}$$

- This rule allows us to rewrite $\forall X \text{ man}(X) \rightarrow \text{mortal}(X)$ plus $\text{man}(\text{socrates})$ into $\text{mortal}(\text{socrates})$
- Chapter 2 of your book gives a whole pile of such inference rules. Over history, philosophers and logicians have identified hundreds of different inference rules.
- It turns out, though, that most of those rewrite rules are equivalent, or you can derive some of them from others of them.
- We'd *like* to use the smallest set of inference rules that we can.
- So the first thing we have to ask is, by what criteria do we judge that a set of inference rules is acceptable?
- There are 2 properties that we normally identify:
 - **Complete:** every theorem that is *true* given your axioms can be *derived* from them with the rewrite rules.
 - **Sound:** you can't derive anything *false* using your rewrite rules (and given a consistent set of axioms, that is).
- So... How many inference rules do we *need* to be both complete and sound for FOPL?
- It turns out that the answer is 1! Somewhat surprisingly, the single rule, **resolution**, can be used to do arbitrary inference in FOPL.
- Resolution is at the heart of most modern automated theorem proving systems and is built-in to Prolog.
- Next time, we'll discuss how resolution works. For the moment, we'll need the tool **unification** to get there.
- The fundamental problem is that in order to apply a rewrite rule, we need to know when two things are the "same".
- E.g., are $\text{plus}(3, 2)$ and $\text{plus}(3, X)$ the same? What about $\text{likes}(X, \text{rabbit})$ and $\text{likes}(\text{Sam}, Y)$? What about $\text{plus}(3, 2)$ and $\text{plus}(2, 3)$?
- Aha! The last two are *not* actually the same! They are *mathematically* equivalent, but we're worried about *syntactic* equivalence at the moment.