CS4450 Lecture 8 Notes

Wednesday, September 5, 2018 3:02 PM

HW1 is posted, it's due this next Monday!

These notes below have a direct bearing on problem 7:

The Definitional Extensions are very useful in defining more connectives.

Defined Connectives as Functions

```
orPL :: Prop -> Prop -> Prop -- Will take in 3 props as inputs and will
define either phi or gamma.
-- Recall that "phi OR gamma" is defined as "NOT phi IMPLIES gamma"
orPL phi gamma = Imply (Not phi) gamma
andPL :: Prop -> Prop -> Prop
-- Recall that "phi AND gamma" is defined as NOT(NOT phi OR NOT gamma)
andPL phi gamma = Neg(OR PL (Neg phi) (Neg gamma)) -- Neg is the same
thing as Not/NOT, as far as I understand.
iffPL :: Prop -> Prop -> Prop
iffPL phi gamma = andPL (Imply phi gamma) (Imply gamma phi)
```

Why do we care about "conservative extension"? It's a very common approach to building a programing langauge, in which you start with a "core" and then build additional features based on the core features, this way when the compiler sees the additional features, it translates them down to the core features and then works with those.

Sometimes, a big language is defined by a pretty small core.

<u>Two Notions of Validity for Propositional Logic</u>

When you say something in logic, you want to know if the statements are true or false, and what do the true or false mean (e.g. what does validity mean)

Truth tables say that you have a "sentence" (e.g. NOT A IMPLIES NOT B), and you want to know "under what conditions is this true or false"? You lay the True and False combinations in a table and go through to calculate the truth value of the things.

Semantics (a.k.a., model theory) is another way of establishing the validity of a wff.

Axiom System for Propositional Logic

What is the Axiom system? There are several axioms, and axioms are things that are always true.

What the axiom system is that it says "You give me a gamma or phi and I'll give you an axiom."

There is only one inference rule in propositional logic, namely "Modus Ponens". What it means is that if phi is true, and if phi implies gamma, then gamma is true.

The inference rule is a lot like function application (e.g. functions in programming).

<u>Instances</u>

An instance of an axiom is substituting phi or gamma with A or B. We'll encounter substitution in a programming languages concept later.

<u>Formal Proofs</u>

Proofs are similar to the derivation systems we looked at with Context-Free Grammars (CFGs).

You can write a proof and verify that it's a proof because you can show that through Modus Ponens or Axioms that it's true.

In certain areas of mathematics, computers are used to prove proofs. It's not purely theoretical.

Proofs will have Ax.n (where n is [1..3]) Or MP next to them, showing which axiom or if Modus Ponus was used to get to that step.

For each axiom scheme, there is an associated function.

```
--Representing the axioms as functions:
axiom1 :: Prop -> Prop -> Prop
axiom1 phi gamma = Imply phi (Imply gamma phi)
-- Axioms as a data type:
-- This is useful because instead of using the axioms as functions, we
can use them as data types. We're doing the opposite approach with what
we did before with orPL, andPL, and iffPL
-- Review on Show by using the axioms as functions:
instance Show Axioms where
show (Ax1 phi gamma) = show (axiom1 phi gamma) -- Here, we're defining
show in terms with show. The fact that we have an equation with two
things that look the same but are different types is hard to understand
at first
show (Ax2 phi gamma psi) = show (axiom2 phi gamma psi)
show (Ax3 phi gamma) = show (axiom3 phi gamma)
-- What about the inference rule, Modus Ponens? We'll define a notion
of theorem.
```

Theorems

A theorem is either an axiom or the modus ponens of two theorems.

Hint: Theorem will be recursive.

Something to think about: Is it possible to define a Theorem value in Haskell that is not a theorem? (Will NOT be on the midterm, is merely to expand your brain!)