CS4450 Lecture 6 Notes

Friday, August 31, 2018 3:00 PM

Today: Representing design in Haskell

HW1 will be out Tuesday, 09/04

Remember: Derivation Systems are used to show how a program does things.

Propositional Logic is Boolean Logic. It's the smallest non-trivial language we can study.

Truth tables are a way to represent strings of symbols as a table of symbols in boolean logic.

A **proposition** is something that's always true or always false.

A **propositional sentence** is a bunch of propositions chained together.

Propositional Logic is the idea that if one propositional sentence holds, will another one hold as well?

A propositional formula can take the form of a propositional variable (p,q,r), a negation (e.g. -L if L is a propositional formula), or an implication.

We use Context Free Grammars to create a Language Syntax

To determine if a particular sequence of symbols is in a language, we perform a derivation of the string. (note, "string" is interchangeable with "sequence of symbols")

"Woofs": Logicians use this to mean "Well-Formed Formulae of Propositional Logic".

Definitional Extensions:

Definition (Disjunction, Conjunction and Equivalence)

> Familiar connectives are defined by existing things in the language. "or" is defined from negation and implies, "and" is defined using OR, and "equivalence" is defined using AND and OR

Since OR can be defined from a negation and cup (implies), we can define OR as such and AND via OR. This allows us to have our definitional extensions built on top of existing things in the language.

Language Syntax as a Haskell data declaration:

```
type Var = String
data Prop = Atom Var
| Not Prop --If we have a Prop, we can make it Not Prop
| Imply Prop Prop -- If we have two Props, we can imply to them and get
the implication.
-- This is a Haskell data type but it is defined by a Context-Free
Grammar
--All of these data declarations can be viewed as Context-Free
Grammars.
```

Example:

```
negp = Not (Atom "p")
-- Compare with: Prop -> (-Prop)
-- -> (-p)
```

This type of stuff will be on the homework!

P V -P (P OR NOT P)

```
-- This is a Haskell data type but it is defined by a Context-Free
Grammar
--All of these data declarations can be viewed as Context-Free
Grammars.
negp = Not (Atom "p")
-- Compare with: Prop -> (-Prop)
-- -> (-p)
or p q = Imply (Not p) q
p OR q = NOT p IMPLIES q
NOT p: (Not(Atom "p"))
p: (Atom "p")
pnp :: Prop
pnp = or (Atom "p") (Not(Atom "p"))
-- No instance for (Show Prop): This doesn't work because there needs
```

```
to be a function called "Show". Basically, Show allows you to "pretty
print" Prop.
-- How to implement:
-- We must write a function called show :: Prop -> String
-- To write a function of a particular type, we must ALWAYS start off
from its type template.
-- - It is almost always the right way to go.
```

What do we do to write show? First of, we'll need to manipulate strings.

String concatenation in Haskell: "hey" ++ "pal" will become "heypal"

The ++ is "append", it concatenates strings together.

Replacing undefined:

HW 1 will be given out Tuesday, it will be a simple homework!