Propositional Logic

Logic and Programming Languages

Lecture #1

Logical Formula

- Domain specific elements
  - propositions
- Connectives (make new from old)
  - And, Or, Not, Implies, etc.
- Judgments about logical formula
  - $\Downarrow$, $=|$, $\cong$ etc
- Proofs
  - A special kind of judgement

Structure

- Syntax
  - Marks on paper
  - Meta-language vs Object-language
  - Meta-language conventions
    - Parentheses, operator precedence, etc.
- Inductive structure
  - Tree like
- Semantics
  - Meaning: usually a function

Inductive Sets

- See the notes 02InductiveSets.pdf

- Pay attention to
  - Trees as inductive sets
  - Languages as inductive sets
  - Recursive definitions
  - Proofs over inductive sets

Comments on Smullyan Chapter 1

- Long discourse on Trees
  - Attempt to formalizing languages as inductive structures
- Lots of discussion about parentheses and other meta language issues
  - "We remark that with this plan, we can (and will) still use parentheses to describe formulas, but the parnetheses are *not* parts of the formula."
- Key result uniqueness of decomposition in order to support and formalize the notion of a proof
- Sub formulas
- Induction principle by induction over the degree (number of connectives) a form of natural number induction.

Propositional Logic

1. A propositional variable is a formula
2. If X is a formula, then ~X is a formula
3. If X and Y are formula, then
    1. $X \wedge Y$ is a formula
    2. $X \vee Y$ is a formula
    3. $X \supset Y$ is a formula (sometimes written as $\Rightarrow$ or $\rightarrow$)

1. Some presentations add T and F are formula

– Note inductive structure, unique decomposition, induction principle.

As a Haskell datatype

```
data Prop a =
     LetterP a
   | AndP (Prop a) (Prop a)
   | OrP (Prop a) (Prop a)
   | ImpliesP (Prop a) (Prop a)
   | NotP (Prop a)
   | AbsurdP
   | TruthP
```

Boolean Valuations

- Introduce two unique elements of the Booleans, True and False (not to be confused with T and F) , sometimes called truth values.

- Function mapping each propositional variable to one of the Boolean elements True or False.

If v is a variable valuation, we often write, for example, v(X) = True, v(Y) = False, etc

Valuation function

- We can lift a valuation function over propositional variables to one over propositional formula.

```
value :: (t -> Bool) -> Prop t -> Bool
value vf TruthP = True
value vf AbsurdP = False
value vf (NotP x) = not (value vf x)
value vf (AndP x y) = (value vf x) && (value vf y)
value vf (OrP x y) = (value vf x) || (value vf y)
value vf (ImpliesP x y) =
   if (value vf x) then (value vf y) else True
value vf (LetterP x) = vf x
```

Interpretations

- An interpretation of a propositional formula is induced by every boolean valuation of its propositional variables.

- Can a formula have more than one interpretation?

Semantics

- We often call the valuation of a formula its semantics.

- Thus we see that the “meaning” of a formula is a function from the valuation of its variables to a Boolean.

Truth tables

- A visual representation of the semantics, usually for formula with two variables.

A	B	$\neg A$	$A \land B$	$A \lor B$	$A \rightarrow B$	$A \leftrightarrow B$
t	**t**	**f**	**t**	**t**	**t**	**t**
t	**f**	**f**	**f**	**t**	**f**	**f**
f	**t**	**t**	**f**	**t**	**t**	**f**
f	**f**	**t**	**f**	**f**	**t**	**t**

Tautology

- A propositional formula is a tautology if its valuation is true for all boolean valuations

- How might we prove if a formula is a tautology?
- How might we write a program to do this?

Satisfiability

- A formula is *satisfiable* if there exists at least one boolean valuation that makes its value True. Sometimes called consistency

- A formula is *unsatisfiable* if no boolean valuation that makes its value True. I.e. it is false under all valuations. Sometimes called inconsistency.

Relations over propositional formula

- Note that tautology, sat, and unsat unary are relations over propositional formula.

```
tautology :  Prop n -> Bool
sat :  Prop n -> Bool
unsat:  Prop n -> Bool
```

Binary relations over formula

- Truth functional equivalence
  - Two formula are true under the exact same set of boolean evaluations (written many ways $\cong, \approx$)
- Truth functional implication
  - Every boolean evaluation that makes S true also makes T true.

- Question? Suppose S and T are functionally equivalent. What can we say about the propositional formula $S \supset T$

Don’t confuse relations with formula

- $A \wedge A \cong A$

- Cannot be $A \wedge (A \cong A)$

Alternate ways to define propositional formula

- There are many ways to define propositional formula because some of the introduction forms ($\sim, \wedge, \vee, \supset$) have functionally equivalent forms using other primitives.

Equivalences as Programs

- Once we know that two formula are equivalent we are justified in replacing one with another if we are only interested in the semantics of the formula.
- We can write programs that transform one formula into an equivalent formula.
- We can often “simplify” formula using equivalences.

Some equivalences

- $x \wedge T \cong x$
- $x \wedge F \cong F$
- …

idempotency laws

$$A \wedge A \simeq A$$
$$A \vee A \simeq A$$

commutative laws

$$A \wedge B \simeq B \wedge A$$
$$A \vee B \simeq B \vee A$$

associative laws

$$(A \wedge B) \wedge C \simeq A \wedge (B \wedge C)$$
$$(A \vee B) \vee C \simeq A \vee (B \vee C)$$

distributive laws

$$A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$$
$$A \wedge (B \vee C) \simeq (A \wedge B) \vee (A \wedge C)$$

de Morgan laws

$$\neg(A \wedge B) \simeq \neg A \vee \neg B$$
$$\neg(A \vee B) \simeq \neg A \wedge \neg B$$

other negation laws

$$\neg(A \rightarrow B) \simeq A \wedge \neg B$$
$$\neg(A \leftrightarrow B) \simeq (\neg A) \leftrightarrow B \simeq A \leftrightarrow (\neg B)$$