Lecture 8: Semantics of Propositional Logic

CS 181O
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

Pig Latin in Haskell

```
pigLatin :: String -> String
pigLatin [] =
pigLatin str = if (isVowel (head str))
            then str ++ yay
                else if (not (isVowel (head(tail(str)))))
                then ((tail(tail str))) ++ [(head str)] ++ [(head(tail str))] ++ ay
                   else (tail str)++[(head str)]++ay

--Check if string is a vowel
isVowel :: Char -> Bool
isVowel v = v `elem` ['a','e','i','o','u']

toPigLatin str = unwords (map pigLatin (words str))
```

Finish Balanced Parens

Balanced Parens

- Every propositional formula F has equal numbers of left and right parentheses. ✔ Moreover every proper prefix of F has more left parentheses then right parentheses.
- *Proof*: Did base case, inductive case left

Balanced Parens (2)

- *Induction step:* S'pose all proper prefixes of **F** have more left than right parens, then show for **(¬F)**.
  - What are proper prefixes of **(¬F)**: **(**, **(¬**, **(¬P**, and **(¬F**
    - Are there more left than right parens in each?
      - Last 2 cases: **(¬P** has more left than right because **P** is a proper prefix of F & therefore has more left than right. **(¬P** has even one more, so fine.
      - **(¬F** We know **F** has same number of left as right, so **(¬F** has one more left than right.
- The cases for **(F∨F)** and **(F∧F)** are similar, but a bit more complex.

Proofs on Propositional Logic

- *Proof in text for Prop 4.3 is incomplete. Can you see why?*
- Structural induction principle should guide you in writing recursive algorithms on formulas of propositional logic.

Semantics of Propositional Logic

- Meaning of formula depends on meaning of propositional letters.
  - Start with valuation fcn V: Prop Letters → {true,false}
  - Extend to V^+: Prop Logic Formulas → {true,false} by
    - $V^+(p) = V(p)$ if p is propositional letter
    - $V^+(\neg\phi)$ = false iff $V^+(\phi)$ = true
    - $V^+(\phi\vee\gamma)$ = true iff $V^+(\phi)$ = true or $V^+(\gamma)$ = true (or both)
    - $V^+(\phi\wedge\gamma)$ = true iff $V^+(\phi)$ = true and $V^+(\gamma)$ = true
    - $V^+(\phi\rightarrow\gamma)$ = false iff $V^+(\phi)$ = true and $V^+(\gamma)$ = false

Truth Tables

P	Q	$\neg P$	$P\wedge Q$	$P\vee Q$	$P\rightarrow Q$	$P\leftrightarrow Q$
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

Each row corresponds to different valuation

Categories of WFFs

- A formula ϕ is *valid*, or a *tautology*, if for all valuations V, we have $V^+(\phi)$ = true.
- A formula ϕ is *satisfiable* if for some valuation V, we have $V^+(\phi)$ = true.
- A formula ϕ is *contingent* if for some valuation V, we have $V^+(\phi)$ = false.
- A formula ϕ is *unsatisfiable*, or a *contradiction*, if for all valuations V , we have $V^+(\phi)$ = false.

Semantic Entailment

- $\phi_1, \ldots, \phi_n \vDash \psi$ iff for every valuation V s.t. $V^*(\phi_1) = \ldots = V^*(\phi_n)$ = true, then $V^*(\psi)$ = true
  - Example: $P \vDash Q \rightarrow P$
  - Read $\phi_1, \ldots, \phi_n \vDash \psi$ as $\phi_1, \ldots, \phi_n$ *logically implies* ψ
- Hence, $\vDash \psi$ iff ψ is a tautology.
- Show: $\phi_1, \ldots, \phi_n, \phi \vDash \psi$ iff $\phi_1, \ldots, \phi_n \vDash \phi \rightarrow \psi$

Propositional Logic

- Definition of well-formed formulas of prop logic:
  - atom := p | q | r | atom'
  - F ::= atom | $(\neg F)$ | $(F \vee F)$ | $(F \wedge F)$

Use ::= in place of → for productions to avoid confusion when expand

- Parens help build unique parse trees for formulas.

Syntax In Haskell

```
data Form =  P String | Ng Form | Cnj [Form] | Dsj [Form]
          deriving Eq

instance Show Form where
 show (P name) = name
 show (Ng  f)  = '-': show f
 show (Cnj fs) = '&': show fs
 show (Dsj fs) = 'v': show fs

form1, form2 :: Form
form1 = Cnj [P p, Ng (P p)]  -- any # args
form2 = Dsj [P p1, P p2, P p3, P p4]
```

From FSynF.hs

Semantics in Haskell

```
-- Find all names in the formula
propNames :: Form -> [String]
propNames (P name) = [name]
propNames (Ng f)   = propNames f
propNames (Cnj fs) = (sort.nub.concat) (map propNames fs)
propNames (Dsj fs) = (sort.nub.concat) (map propNames fs)

-- Generate all valuation for given names
genVals :: [String] -> [[(String,Bool)]]
genVals [] = [[]]
genVals (name:names) = map ((name,True) :) (genVals names)
                 ++ map ((name,False):) (genVals names)

-- List of all possible valuations for atoms in formula
allVals :: Form -> [[(String,Bool)]]
allVals = genVals . propNames
```

Semantics in Haskell

```
-- eval takes valuation and formula and gives value
eval :: [(String,Bool)] -> Form -> Bool
eval [] (P c)    = error (no info about  ++ show c)
eval ((i,b):xs) (P c)
    | c == i    = b
    | otherwise = eval xs (P c)

eval xs (Ng f)   = not (eval xs f)
eval xs (Cnj fs) = all (eval xs) fs
eval xs (Dsj fs) = any (eval xs) fs
```

From FSemF.hs

Semantics in Haskell

```
- - Is formula a tautology or satisfiable or a contradiction
tautology :: Form -> Bool
tautology f = all (\ v -> eval v f) (allVals f)

satisfiable :: Form -> Bool
satisfiable f = any (\ v -> eval v f) (allVals f)

contradiction :: Form -> Bool
contradiction = not . satisfiable

- - Does first formula logically imply second
implies :: Form -> Form -> Bool
implies f1 f2 = contradiction (Cnj [f1,Ng f2])
-- If start with list of vals and formula F then returns sublist
-- making F true
update :: [[(String,Bool)]] -> Form -> [[(String,Bool)]]
update vals f = [ v | v <- vals, eval v f ]
```

Predicate Logic