Lecture 10: Semantics of Predicate Logic

CS 181O
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

Quick Review

Substitution

- Define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.
  - Expect $\forall x.\phi(x) \Rightarrow \phi[t/x]$ for every term t
  - What about $\forall x.\exists y.L(x,y) \Rightarrow \exists y.L(y,y)$?

More Substitution

- Say that *t is free for x in ϕ* if no free x leaf in ϕ occurs in the scope of $\forall y$ or $\exists y$ for any variable y occurring in t.
  - y not free for x in $\exists y\ L(x,y)$
  - Only allow substitution $\phi[t/x]$ if t free for x in ϕ
  - If t not free for x in ϕ, rename bound variables to make substitution legal.

Typed Predicate Calculus

- Variant where bound variables have types
  - $\exists$x: T, $\forall$y: U
- Examples:
  - Fido bit someone $\Rightarrow$ $\exists x$: Person. B(f, x))
  - Every dog bites Sally $\Rightarrow$ $\forall$x: Dog. B(x,s))
  - Some dog bit Sally $\Rightarrow$ $\exists$x: Dog. B(x,s))
- Can be translated away

Semantics

- More complex than for propositional logic.
- Must interpret all the terms as elements of a model and determine what tuples satisfy which relation.
  - Then can build up meaning as before with $\neg$, $\wedge$, and $\vee$.
- Quantifiers tricker.

Semantics

- A model $\mathcal{M} = (D,I)$ for a predicate logic has the following components:
  - Non-empty set D called the domain of the model $\mathcal{M}$.
  - For each constant symbol c, there is an element I(c) of D.
  - For each k-ary function symbol f , there is a function $I(f) : D^k \to D$.
  - For each k-ary predicate symbol P, there is a subset I(P) of D^k.

Semantics

- How do we interpret wff: $P(x,y) \wedge \forall x Q(x,x,y)$.
- Interpret over domain D of real numbers, with $I(P) = \{(x,y) \mid x<y\}$ and $I(Q) =\{(u, v, w) \mid u = v + w\}$.
- $\mathcal{M} \models \forall y\ (\exists x\ Q(x, x, y) \to \forall x\ Q(x, x, y))$
- What about free variables? Need g: var $\to D$ *lookup table*
- Then $\mathcal{M}, g \models P(x, y) \wedge \forall x\ Q(x, x, y)$ if and only if $g(x) < 0$ and $g(y) = 0$.

Defining Truth!

- Due to my (*academic*) grandfather: Alfred Tarski in 1933. Cleaned up in 1956.
- Must separate meta-language from the language studying.
- Want compositional meaning:
  - Meaning of whole depends on meaning of parts.
- Notation: If g: var $\to D$, x is a vble, and a $\in D$, define $g[x := a](y) = g(y)$ if $y \neq x$
  $= a$ if $y = x$

Meanings of terms

- Given a model $\mathcal{M} = (D,I)$, define meaning of terms with respect to g inductively as follows:
  - $g_{I,g}(x) = g(x)$ for x a variable
  - $g_{I,g}(c) = I(c)$ for c a constant
  - $g_{I,g}(f(t1,\ldots,tk)) = I(f)\ (g_{I,g}(t_1),\ldots, g_{I,g}(t_k))$

Satisfaction

- $\mathcal{M}, g \models P(t_1,\ldots,t_k)$ iff $(g_{I,g}(t_1),\ldots, g_{I,g}(t_k)) \in I(P)$
- $\mathcal{M}, g \models t{=}u$ iff $g_{I,g}(t) = g_{I,g}(u)$
- $\mathcal{M}, g \models \neg\phi$ iff $\mathcal{M}, g \not\models \phi$.
- $\mathcal{M}, g \models \phi \wedge \psi$ iff $\mathcal{M}, g \models \phi$ and $\mathcal{M}, g \models \psi$.
- $\mathcal{M}, g \models \phi \vee \psi$ iff $\mathcal{M}, g \models \phi$ or $\mathcal{M}, g \models \psi$.
- $\mathcal{M}, g \models \phi \to \psi$ iff $\mathcal{M}, g \not\models \phi$ or $\mathcal{M}, g \models \psi$.
- $\mathcal{M}, g \models \exists x\ \phi$ iff for some $a \in D$, $\mathcal{M}, g[x := a] \models \phi$.
- $\mathcal{M}, g \models \forall x\ \phi$ iff for all $a \in D$, $\mathcal{M}, g[x := a] \models \phi$.
- *Compositional meaning!!*

Properties of Semantics

- Prop: If g and g' agree on all the free variables in ϕ, then $\mathcal{M}, g \models \phi$ if and only if $\mathcal{M}, g' \models \phi$.
- A wff without free variables is called a *sentence*.
- Prop: If ϕ is a sentence, then either $\mathcal{M}, g \models \phi$ for all g or $\mathcal{M}, g \models \neg\phi$ for all g, but not both.
  - So just write $\mathcal{M} \models \phi$ if ϕ is a sentence.

Examples

- Let $\mathcal{M}$ have domain $D = \{x \in Q \mid 0 \le x \le 1\}$, $I(LT) = \{(q,r) \mid q < r\}$. $g(x) = 0$, $g(z) = 1/2$. Then
- $\mathcal{M}, g \models \forall z\, (LT(x,z) \vee x = z)$
- $\mathcal{M} \models \exists y\, \forall z\, (LT(y,z) \vee y = z)$

Satisfiability

- The set Γ is *satisfiable* if exists a model $\mathcal{M}$ and environment g such that $\mathcal{M}, g \models \gamma$ for all $\gamma \in \Gamma$.
- A formula ϕ is *valid* if, for all models $\mathcal{M}$ and environments g, $\mathcal{M}, g \models \phi$.
- $\Gamma \models \psi$ (*read* Γ *semantically entails* ψ) iff ψ is true in every model and environment which make all the formulas of Γ true.
- ϕ and ψ are logically equivalent iff ???

Satisfiability Example

- Does $\forall x\, \neg\phi \models \neg\forall x\, \phi$
  - Reverse?

Predicate Logic in Haskell

Defining Variables:

```
type Name     = String
type Index    = [Int]
data Variable = Variable Name Index deriving (Eq,Ord)

instance Show Variable where
  show (Variable name [])  = name
  show (Variable name [i]) = name ++ show i
  show (Variable name is ) = name ++ showInts is
    where showInts []     = ""
          showInts [i]    = show i
          showInts (i:is) = show i ++ "_" ++ showInts is
```

From FSynF.hs

Predicate Logic in Haskell

```
x, y, z :: Variable
x = Variable "x" []
y = Variable "y" []
z = Variable "z" []

data Formula a = Atom String [a]          type a is type of terms
             | Eq a a
             | Neg  (Formula a)
             | Impl (Formula a) (Formula a)
             | Equi (Formula a) (Formula a)    throw in extra connective
             | Conj [Formula a]
             | Disj [Formula a]
             | Forall Variable (Formula a)
             | Exists Variable (Formula a)
             deriving Eq
```

```
instance Show a => Show (Formula a) where
  show (Atom s [])   = s
  show (Atom s xs)   = s ++ show xs
  show (Eq t1 t2)    = show t1 ++ "==" ++ show t2
  show (Neg form)    = '~' : (show form)
  show (Impl f1 f2)  = "(" ++ show f1 ++ "==>"
                       ++ show f2 ++ ")"
  show (Equi f1 f2)  = "(" ++ show f1 ++ "<=>"
                       ++ show f2 ++ ")"
  show (Conj [])     = "true"
  show (Conj fs)     = "conj" ++ show fs
  show (Disj [])     = "false"
  show (Disj fs)     = "disj" ++ show fs
  show (Forall v f)  = "A " ++  show v ++ (' ' : show f)
  show (Exists v f)  = "E " ++  show v ++ (' ' : show f)
```

Sample Formulas

All of type Formula Variable

```
formula0 = Atom "R" [x,y]
formula1 = Forall x (Atom "R" [x,x])
formula2 = Forall x
             (Forall y
               (Impl (Atom "R" [x,y]) (Atom "R" [y,x])))
```

Adding Terms

```
data Term = Var Variable | Struct String [Term]
      deriving (Eq,Ord)

instance Show Term where
  show (Var v)        = show v
  show (Struct s []) = s
  show (Struct s ts) = s ++ show ts
                                          constants are 0-ary functions
tx, ty, tz, one, two, sum :: Term
tx = Var x                 one = Struct "1" []
ty = Var y                 two = Struct "2" []
tz = Var z                 sum = Struct "Plus" [one,two]

        simple :: Formula Term
        simple = Eq sum two
```

Questions?