Lecture 11: Language to Logic

CS 181O
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

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
```

type a is type of terms

throw in extra connective

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

tx, ty, tz, one, two, sum :: Term
tx = Var x                  one = Struct "1" []
ty = Var y                  two = Struct "2" []
tz = Var z                  sum12 = Struct "Plus" [one,two]

        simple :: Formula Term
        simple = Eq sum12 two
```

constants are 0-ary functions

Formulas with terms

```
simple = Eq sum12 two

univ = Forall x (Eq tx two)

eqTest = Forall x (Forall y(Eq sumxy two))

-- relation LessThan on one, two
reln = Atom "LessThan" [one, two]
```

Hold off implementing semantics for Predicate Logic

Interpreting language

- Two options: indirect & direct

Natural language expression

Logical representation

Model-theoretic interpretation

Indirect First

- A challenge:
  - "Fido likes a bone" *translates as*
    - $\exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(f,b))$
    - Translate "likes a bone" as $\lambda d.\ \exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(d,b))$ then $(\lambda d.\ \exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(d,b)))(f)$ gives final meaning (*where f is Fido)*
  - Compositional
    - Look at parse trees: apply predicate to subject

Indirect First

- Quantifiers look problematic:
  - "Fido likes a bone" *translates as*
    - $\exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(f,b))$
    - Translate "likes a bone" as $\lambda d.\ \exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(d,b))$ then $(\lambda d.\ \exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(d,b)))(f)$ gives final meaning
  - "Every dog likes a bone" translates as
    - $\forall d.\ (\mathrm{Dog}(d) \rightarrow \exists b.\ (\mathrm{Bone}(b) \wedge \mathrm{Likes}(d,b)))$
- Is this compositional?
  - How does this come from "likes a bone" and "every dog"?
  - Look at parse trees of original and translation

Solution

- Replace individual by the set of all properties they satisfy.
  - Ex. Instead of constant fido, represent as $\{P \mid P(fido)\}$
    - Though of course use characteristic function instead: $\lambda P.\ P(fido)$
  - Every dog: $\lambda P.\forall d.(Dog(d) \to P(d))$
  - Likes a bone: $\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b))$

Solution

- "Fido": $\lambda P.\ P(fido)$
- "Every dog": $\lambda P.\forall d.(Dog(d) \to P(d))$
- "Likes a bone": $\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b))$
- Fido likes a bone: $\Rightarrow$
  - $(\lambda P.\ P(fido))(\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b))) =_\beta$
    $(\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b)))(fido) =_\beta$
    $\exists b.\ (Bone(b) \wedge Likes(fido,b)))$
- Every dog likes a bone $\Rightarrow$
  - $(\lambda P.\forall d.(Dog(d) \to P(d)))(\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b))) =_\beta$
    $\forall d.\ (Dog(d) \to (\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x,b)))(d)) =_\beta$
    $\forall d.\ (Dog(d) \to \exists b.\ (Bone(b) \wedge Likes(d,b)))$

Quantifiers

- Other quantifiers:
  - Likes a bone: $\lambda x.\ \exists b.\ (Bone(b) \wedge Likes(x, b))$
  - A dog: $\lambda P.\ \exists d.\ (Dog(d) \wedge P(d))$
  - A dog likes a bone?
- So what is meaning of "a" or "all" or "the"?
  - "a" $\Rightarrow \lambda Q.\ \lambda P.\ \exists x.\ (Q(x) \wedge P(x))$
  - "every" $\Rightarrow \lambda Q.\ \lambda P.\ \forall x.\ (Q(x) \to P(x))$
  - "the" $\Rightarrow \lambda P\ \lambda Q \exists x \forall y[[P\ (y) \leftrightarrow x = y] \wedge Q(x)]$

Types

- Let e be type of elements in universe, t be truth values.
- Type of sentence is t
- Types of Determiners?
  - "a" $\Rightarrow \lambda Q: e \to t.\ \lambda P: e \to t.\ \exists x: e.\ (Q(x) \wedge P(x))$
  - "every" $\Rightarrow \lambda Q: e \to t.\ \lambda P: e \to t.\ \forall x: e.\ (Q(x) \to P(x))$
  - "the" $\Rightarrow \lambda P: e \to t.\ \lambda Q: e \to t.$
    $\exists x: e.\ \forall y: e.\ ((P\ (y) \leftrightarrow x = y) \wedge Q(x))$
- All have type $(e \to t) \to (e \to t) \to t$

More types:

- Type of determiner: $(e \to t) \to (e \to t) \to t$
- Type of Noun?
  - $e \to t$
- Type of noun phrase?
  - $(e \to t) \to t$
- Type of verb phrase?
  - $e \to t$
- Type of sentence? t!

Natural Language Semantics

- Take grammar from lecture 6 and translate sentences to predicate logic.
- Use lambda calculus for semantics of phrases.
- Compose using function application.
- Meaning of sentence is formula of predicate logic.

Grammar from lecture 6

- S → NP VP
- NP → Snow White | Alice | Dorothy | Goldilocks | DET CN | DET RCN
- DET → the | every | some | no
- CN → girl | boy | princess | dwarf | giant | sword | dagger
- RCN → CN that VP | CN that NP TV
- VP → laughed | cheered | shuddered | TV NP | DV NP NP
- TV → loved | admired | helped | defeated | caught
- DV → gave

Syntax

- Review FSynF.hs

```
data Term = Var Variable | Struct String [Term]
          deriving (Eq,Ord)
data Formula a = Atom String [a]
               | Eq a a
               | Neg  (Formula a)
               | Impl (Formula a) (Formula a)
               | Equi (Formula a) (Formula a)
               | Conj [Formula a]
               | Disj [Formula a]
               | Forall Variable (Formula a)
               | Exists Variable (Formula a)
               deriving Eq
```

Translating

```
type LF = Formula Term

lfSent :: Sent -> LF
lfSent (Sent np vp) = (lfNP np) (lfVP vp)

lfNP :: NP -> (Term -> LF) -> LF
lfNP SnowWhite     = \ p -> p (Struct "SnowWhite"  [])
lfNP Alice         = \ p -> p (Struct "Alice"      [])
lfNP Dorothy       = \ p -> p (Struct "Dorothy"    [])
lfNP Goldilocks    = \ p -> p (Struct "Goldilocks" [])
lfNP LittleMook    = \ p -> p (Struct "LittleMook" [])
lfNP Atreyu        = \ p -> p (Struct "Atreyu"     [])
lfNP (NP1 det cn)  = (lfDET det) (lfCN cn)
lfNP (NP2 det rcn) = (lfDET det) (lfRCN rcn)
```

Translating

```
lfVP :: VP -> Term -> LF
lfVP Laughed   = \ t -> Atom "laugh"   [t]
lfVP Cheered   = \ t -> Atom "cheer"   [t]
lfVP Shuddered = \ t -> Atom "shudder" [t]

lfVP (VP1 tv np) =
   \ subj -> lfNP np (\ obj -> lfTV tv (subj,obj))
lfVP (VP2 dv np1 np2) =
   \ subj -> lfNP np1 (\ iobj -> lfNP np2 (\ dobj ->
                   lfDV dv (subj,iobj,dobj)))

lfTV :: TV -> (Term,Term) -> LF
lfTV Loved    = \ (t1,t2) -> Atom "love"   [t1,t2]
lfTV Admired  = \ (t1,t2) -> Atom "admire" [t1,t2]
lfTV Helped   = \ (t1,t2) -> Atom "help"   [t1,t2]
lfTV Defeated = \ (t1,t2) -> Atom "defeat" [t1,t2]
```

Translating

```
lfDV :: DV -> (Term,Term,Term) -> LF
lfDV Gave = \ (t1,t2,t3) -> Atom "give" [t1,t2,t3]

lfCN :: CN -> Term -> LF
lfCN Girl     = \ t -> Atom "girl"     [t]
lfCN Boy      = \ t -> Atom "boy"      [t]

lfCN Princess = \ t -> Atom "princess" [t]
lfCN Dwarf    = \ t -> Atom "dwarf"    [t]
lfCN Giant    = \ t -> Atom "giant"    [t]
lfCN Wizard   = \ t -> Atom "wizard"   [t]
lfCN Sword    = \ t -> Atom "sword"    [t]
lfCN Dagger   = \ t -> Atom "dagger"   [t]
```

Translating

- Rather than continuing to paste code here, I'll just refer you to the file MCWPL.hs
- Notice that there are two evaluation functions. One, eval, just interprets formulas that involve variables, while the other, evl, evaluates formulas that involve function symbols as well.

Questions?