Model Theoretic Semantics

Vocabulary

data Vocab =
  V0 Thing
  | V1 Rel1
  | V2 Rel2

data Thing =
  T Bart
  | T Lisa
  | T SantaLH

data Rel1 =
  U Student
  | U Greyhound
  | U Run

data Rel2 =
  B Sibling
  | B Torment

data Entity = E1 | E2 | E3 |..

deriving (Bounded)

type Model = Vocab → {Entity}

model9871 :: Model
model9871 (V0 Bart)          = {E1}
model9871 (V0 Lisa)          = {E2}
model9871 (V0 SantaLH)       = {E3}
model9871 (V1 (U Student))   = {E1, E2}
model9871 (V1 (U Greyhound)) = {E3}
model9871 (V1 (U Runs))      = {E2, E3}
model9871 (V2 (B Sibling))   = {(E1, E2), (E2, E1)}
model9871 (V2 (B Torment))   = {(E1, E2)}

Assignment

type Assignment = Variable → Maybe Entity

assign3193 :: Assignment
assign3193 (Variable "x") = Just E1
assign3193 (Variable "y") = Just E2
assign3193 _ = Nothing

Language

newtype Variable =
  Variable String

data Term =
  Const Thing
  | Var Variable

data Formula =
  UnaryRel Rel1 Term
  | BinaryRel Rel2 Term Term
  | Not Formula
  | And Formula Formula
  | Or Formula Formula
  | Implies Formula Formula
  | Exists Variable Formula
  | ForAll Variable Formula

data Formula2
  Lambda Variable Formula
  | Apply Variable Formula Formula2
  | NForm Formula

Interpretation

class Interp res v where
  interp :: Model → Assignment → v → Maybe res

instance Interp Entity Variable where
  interp mdl asg v = asg v

instance Interp Entity Vocab where
  interp mdl asg v = Just (mdl v)

instance Interp Entity Thing where
  interp mdl asg t = interp mdl asg (V0 t)

instance Interp Entity Term where
  interp mdl asg (Constant c) = interp mdl asg c
  interp mdl asg (Var v) = intern mdl v

instance Interp Bool Formula where
  interp mdl asg f = satisfies mdl asg f

satisfies :: Model → Assignment → Formula → Maybe Bool
satisfies mdl asg f = case f of
  UnaryRel r t -> ⟨ int t ∈ int r ⟩
  BinaryRel r t1 t2 -> ⟨ (int t1, int t2) ∈ int r ⟩
  Not f -> ⟨ not (sat f) ⟩
  And f1 f2 -> ⟨ sat f1 && sat f2 ⟩
  Or f1 f2 -> ⟨ sat f1 || sat f2 ⟩
  Implies p q -> ⟨ not (sat f1) || sat f2 ⟩
  Exists x f -> ⟨ any (\a -> satisfies mdl a f) \ variants asg v ⟩
  ForAll x f -> ⟨ all (\a -> satisfies mdl a f) \ variants asg v ⟩

where
  int = interp mdl asg
  sat = satisfies mdl asg f

variants :: Assignment → Variable → [Assignment]
variants asg v0 =
  [ \v -> if v == v0 then Just e else asg v |
    e <- [minBound..maxBound] ] -- for all entities
Something I don't say explicitly in the talk is that we think here of our model as having two types of objects in its domain: Entities and Truth Conditions.

Montague's Intensional Logic has more stuff: Senses, Points in Time, Worlds.

type Rel1 = Entity → Bool

type Rel2 = Entity → Entity → Bool

Lisa thinks :: Bool
Lisa inspires a student :: Bool

Lisa :: Rel1 → Bool
a student :: Rel1 → Bool

thinks :: Rel1
inspires a student :: Rel1

inspires :: Entity → Rel1

Example derivation