A first example

Lexicon

Kathy, NP : kathy
Fong, NP : fong
respects, V : λy.λx. respect(x, y)
runs, V : λx. run(x)

Grammar

S : β(α) → NP : α
VP : β(α) → V : β
NP : α

1. Kathy respects Fong
   VP : respect(Fong)
   = [λx. respect(x, fong)](kathy)
   = respect(kathy, fong)

Database/knowledgebase interfaces

• Assume that respect is a table Respect with two fields respecter and respected
• Assume that kathy and fong are IDs in the database: k and f
• If we assert Kathy respects Fong we might evaluate the form respect(fong)(kathy) by doing an insert operation:
  insert into Respects(respecter, respected) values (k, f)

Typed λ calculus (Church 1940)

• Everything has a type (like Java!)
• Bool truth values (0 and 1)
  Ind individuals
  Ind → Bool properties
  Ind → Ind → Bool binary relations
• kathy and fong are Ind
• run is Ind → Bool
• respect is Ind → Ind → Bool
• Types are interpreted right associatively.
• We convert a several argument function into embedded unary functions. Referred to as currying.
Typed \( \lambda \) calculus (Church 1940)

- Once we have types, we don’t need \( \lambda \) variables just to show what arguments something takes, and so we can introduce another operation of the \( \lambda \) calculus:
  - \( \eta \) reduction [abstractions can be contracted]
    \( \lambda x. (P(x)) \Rightarrow P \)
  - This means that instead of writing:
    \( \lambda y. \lambda x. \text{respect}(x,y) \)
  - we can just write:
    \( \text{respect} \)

Types of major syntactic categories

- nouns and verb phrases will be properties (\( \text{Ind} \rightarrow \text{Bool} \))
- noun phrases are \( \text{Ind} \) – though they are commonly type-raised to (\( \text{Ind} \rightarrow \text{Bool} \)) \( \rightarrow \text{Bool} \)
- adjectives are (\( \text{Ind} \rightarrow \text{Bool} \)) \( \rightarrow \text{Ind} \rightarrow \text{Bool} \)
  
  This is because adjectives modify noun meanings, that is properties.
- Intensifiers modify adjectives: e.g. very in a very happy camper, so they’re ((\( \text{Ind} \rightarrow \text{Bool} \)) \( \rightarrow \text{Ind} \rightarrow \text{Bool} \)) \( \rightarrow ((\text{Ind} \rightarrow \text{Bool} \rightarrow \text{Ind} \rightarrow \text{Bool}) \) [honesti].

A grammar fragment

- \( S : \beta(\alpha) \rightarrow \text{NP} : \alpha \quad \text{VP} : \beta \)
- \( \text{NP} : \beta(\alpha) \rightarrow \text{Det} : \beta \quad \text{N'} : \alpha \)
- \( \text{N'} : \beta(\alpha) \rightarrow \text{Adj} : \beta \quad \text{N'} : \alpha \)
- \( \text{N'} : \beta(\alpha) \rightarrow \text{N} : \alpha \quad \text{PP} : \beta \)
- \( \text{N} : \beta \rightarrow \text{N} : \beta \)
- \( \text{VP} : \beta(\alpha) \rightarrow \text{V} : \beta \quad \text{NP} : \alpha \)
- \( \text{VP} : \beta(\alpha) \rightarrow \text{V} : \beta \quad \text{NP} : \alpha \quad \text{NP} : \gamma \)
- \( \text{VP} : \beta(\alpha) \rightarrow \text{VP} : \alpha \quad \text{PP} : \beta \)
- \( \text{VP} : \beta \rightarrow \text{V} : \beta \)
- \( \text{PP} : \beta(\alpha) \rightarrow \text{P} : \beta \quad \text{NP} : \alpha \)

A grammar fragment

- Kathy, NP : kathy\text{Ind}
- Fong, NP : fong\text{Ind}
- Palo Alto, NP : paloalto\text{Ind}
- car, N : car\text{Ind} \rightarrow \text{Bool}
- overpriced, Adj : overpriced(\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool})
- outside, PP : outside(\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool})
- red, Adj : \lambda P. ((\text{Ind} \rightarrow \text{Bool})) \rightarrow (\text{Ind} \rightarrow \text{Bool}) \rightarrow ((\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool}))
- the, Det : \lambda
- a, Det : some(\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool}) \rightarrow \text{Bool}
- runs, V : run\text{Ind} \rightarrow \text{Bool}
- respects, V : respect(\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool})
- likes, V : like(\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool})
Adjective and PP modification

- \( N' : \lambda x. (\text{car}(x) \land \text{in'}(\text{paloalto})(x) \land \text{red'}(x)) \)

- \( \text{Adj} : \lambda P. (\lambda x. (P(x) \land \text{red'}(x))) \)

- \( \text{red} N' : \lambda x. \text{car}(x) \land \text{in'}(\text{paloalto})(x) \land \text{red'}(x) \)

- \( N' : \lambda x. (\text{car}(x) \land \text{red'}(x)) \)

- \( \text{PP} : \lambda P. \lambda x. (P(x) \land \text{in'}(\text{paloalto})(x)) \)

- \( \text{Adj} : \lambda P. (\lambda x. P(x) \land \text{red'}(x)) \)

- \( \text{red} N : \text{car} \)

- \( \text{PP} : \lambda P. \lambda x. (P(x) \land \text{in'}(\text{paloalto})(x)) \)

Why things get more complex

- When doing predicate logic did you wonder why:
  - Kathy runs
  - no kid runs

- Somehow the NP's meaning is wrapped around the predicate

- Or consider why this argument doesn't hold:
  - Nothing is better than a life of peace and prosperity.

  A cold egg salad sandwich is better than nothing.

  A cold egg salad sandwich is better than a life of peace and prosperity.

  The problem is that nothing is a quantifier

Generalized Quantifiers

- We have a reasonable semantics for red car in Palo Alto as a property from \( \text{Ind} \rightarrow \text{Bool} \)

- How do we represent noun phrases like the red car in Palo Alto or every red car in Palo Alto?

  \[ [[\text{the}])(P) = a \text{ if } (P(b) = 1 \text{ iff } b = a) \]

  undefined, otherwise

  The semantics for the following Bertrand Russell, for whom the \( x \) meant the unique item satisfying a certain description

Generalized Quantifiers

- red car in Palo Alto

  select Cars.obj from Cars, Locations, Red where
  Cars.obj = Locations.obj AND
  Locations.place = 'paloalto' AND Cars.obj = Red.obj
  (here we assume the unary relations have one field, obj).
Generalized Quantifiers

- **the red car in Palo Alto**
  - \( \text{NP} : \iota(\lambda x. \text{car}(x) \land \text{in}(\text{paloalto})(x) \land \text{red}'(x)) \)
  - \( \text{Det} : \iota \)

- **the red car in Palo Alto**
  - \( \text{select Cars.obj from Cars, Locations, Red where} \)
  - \( \text{Cars.obj} = \text{Locations.obj AND Locations.place} = \text{‘paloalto’ AND Cars.obj} = \text{Red.obj having count(*)} = 1 \)

**Representing proper nouns with quantifiers**

- The central insight of Montague’s PTQ was to treat individuals as of the same type as quantifiers (as type-raised individuals):
  - **Kathy**: \( \lambda P. P(\text{kathy}) \)
- Both good and bad
- The main alternative (which we use) is flexible *type shifting* – you raise the type of something when necessary.

**Nominal type shifting**

- Common patterns of nominal type shifting
  - \( \text{Q} \)
    - \( \text{Ind} \)
      - \( \text{R} \)
        - \( \text{some}^2 \)
          - \( (\text{Ind} \rightarrow \text{Bool}) \rightarrow \text{Bool} \)
          - \( \text{R}(x) = \lambda P. P(x) \)
          - \( \text{some}^2(P) = \lambda Q. (Q \cap P) \neq \emptyset \)
          - \( Q(x) = \lambda y. x = y' \)
          - In this diagram, \( R \) is exactly this basic type-raising function for individuals.
Noun phrase scope – following Hendriks (1993)

Value raising raises a function that produces an individual to one that produces a quantifier. If $\alpha : \sigma \rightarrow \text{Ind}$ then $\lambda x.\lambda p.P(\alpha(x)) : \sigma \rightarrow (\text{Ind} \rightarrow \text{Bool}) \rightarrow \text{Bool}$

Argument raising replaces an argument of a boolean function with a variable and applies the quantifier semantically binding the replacing variable. If $\alpha : \sigma \rightarrow \text{Ind} \rightarrow \tau \rightarrow \text{Bool}$ then $\lambda x_1.\lambda q.\lambda x_3.Q(\lambda x_2.\alpha(x_1)(x_2)(x_3)) : \sigma \rightarrow (\text{Ind} \rightarrow \text{Bool}) \rightarrow \text{Bool} \rightarrow \tau \rightarrow \text{Bool}$

Argument lowering replaces a quantifier in a boolean function with an individual argument, where the semantics is calculated by applying the original function to the type raised argument. If $\alpha : \sigma \rightarrow ((\text{Ind} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \tau \rightarrow \text{Bool}$ then $\lambda x_1.\lambda x_2.\alpha(x_1)(\lambda p.P(x_2))(x_3) : \sigma \rightarrow \text{Ind} \rightarrow \tau \rightarrow \text{Bool}$

Some kid broke every toy

- $S : \text{every}^2(\text{toy}((\lambda y,\text{some}^2(\text{kid})(\lambda x,\text{break}(y)(x)))))$

Every student runs

- $S : \text{every}^2(\text{student})(\lambda x,\text{run}(x))$

Some kid broke every toy

- $S : \text{some}^2(\text{kid})$

Questions with answers!

- A yes/no question (Is Kathy running?) will be something of type $\text{Bool}$, checked on database
- A content question (Who likes Kathy?) will be an open proposition, that is something semantically of the type $\text{property} (\text{Ind} \rightarrow \text{Bool})$, and operationally we will consult the database to see what individuals will make the statement true.
- We use a grammar with a simple form of gap-threading for question words

Syntax/semantics for questions

- $S' : \beta(\alpha) \rightarrow \text{NP}[\text{wh}] : \beta$ Aux $S : \alpha$
- $S' : \alpha \rightarrow \text{Aux} S : \alpha$
- $\text{NP}/\text{NP}_z : z \rightarrow e$
- $S : \lambda z.F(\ldots z \ldots) \rightarrow S/\text{NP}_z : F(\ldots z \ldots)$
Syntax/semantics for questions

- who, NP[wh] : A U A x U (x) \land human(x)
  what, NP[wh] : A U U

how, many, Det[wh] : A P . A V . A x . P (x) \land V (x)

- Where | | is the operation that returns the cardinality of a set (count).

Question examples

**S'** : λz. like(z)(kathy)
NP[wh] : M U A x U (x) \land human(x) Aux
Who does S : λz. like(z)(kathy)
S/NP : like(z)(kathy)
NP : Kathy
V : like
Which
N : car
cars
Det : λP . λV . λx . P (x) \land V (x)
Did : λP . A V . A x . P (x) \land V (x)
Which
N : car
cars

- select liked from Likes where Likes.liker='Kathy'

- select liked from Likes where Likes.liker='Kathy' AND Humans.obj = Likes.liked

- select liked from Likes where Likes.liker='Kathy'

Question examples

**S'** : λz. every^2(student)(like(z))
NP[wh] : M U A x U (x) \land V (x) Aux
Who does S : λz. every^2(student)(like(z))
S/NP : every^2(student)(like(z))
NP : Kathy
V : like
Which
N : car
cars
Det : λP . A V . A x . P (x) \land V (x)
Did : λP . A V . A x . P (x) \land V (x)
Which
N : car
cars

- select count(*) from Likes, Cars, Locations, Reds where Cars.obj = Likes.liked AND Likes.liker = 'Kathy' AND Red.obj = Likes.liked AND Locations.place = 'Palo Alto' AND Locations.obj = Likes.liked

- select 'yes' where Seeings.seer = k AND Seeings.seen = (select Cars.obj from Cars, Locations, Red where Cars.obj = Locations.obj AND Locations.place = 'paloalto' AND Cars.obj = Red.obj having count(*) = 1)
How many red cars in Palo Alto does Kathy like?

\[ S : \text{see}(\text{i}(\lambda x. \text{car}(x) \land \text{in}(\text{paloalto})(x) \land \text{red}'(x)))((\text{kathy})) \]

\[ \text{Aux} \quad S \quad \text{does} \quad \text{NP} : \text{kathy} \quad NP : \text{see}(\text{i}(\lambda x. \text{car}(x) \land \text{in}(\text{paloalto})(x) \land \text{red}'(x))) \]

How could we learn such representations?

- After disengagement for many years, there has started to be very interesting work in this area:


How can we reason with such representations?

- Logical reasoning is practical for certain domains (business rules, legal code, etc.) and has been used (see Blackburn and Bos 2005 for background).

- But our knowledge of the world is in general incomplete and uncertain.

- There is various recent work on handling restricted fragments of first order logic in probabilistic models


How can we reason with such representations?

- Undirected model:


  - A recent attempt to apply this to natural language inference:


    - Logical formulae are given weights which are grounded out in an undirected markov network.