## A first example

### Lexicon

- **Kathy, NP:** kathy
- **Fong, NP:** fong
- **respects, V:** $\lambda y. \lambda x. \text{respect}(x, y)$
- **runs, V:** $\lambda x. \text{run}(x)$

### Grammar

- **S:** $\beta(\alpha) \rightarrow NP: \alpha \quad VP: \beta$
- **VP:** $\beta(\alpha) \rightarrow V: \beta \quad NP: \alpha$
- **VP:** $\beta \rightarrow V: \beta$
A first example

• \( S : \text{respect}(\text{kathy}, \text{fong}) \)

• \([\text{VP respects Fong}] : \left[ \lambda y.\lambda x.\text{respect}(x, y) \right] (\text{fong})\)
  \[= \lambda x.\text{respect}(x, \text{fong}) \quad [\beta \text{ red.}]\]

• \([S \text{ Kathy respects Fong}] : \left[ \lambda x.\text{respect}(x, \text{fong}) \right] (\text{kathy})\)
  \[= \text{respect}(\text{kathy}, \text{fong})\]
Database/knowledgebase interfaces

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

• Below we focus on questions like *Does Kathy respect Fong* for which we will use the relation to ask:

  select ‘yes’ from Respects where Respects.respecter = $k$ and Respects.respected = $f$

• We interpret “no rows returned” as ‘no’ = 0.
Typed $\lambda$ 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.
  - **respect** is **Ind → (Ind → Bool)**
- 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}
Typed $\lambda$ calculus (Church 1940)

- $\lambda$ extraction allowed over any type (not just first-order)
- $\beta$ reduction [application]
  \[(\lambda x. P(\cdots, x, \cdots))(Z) \Rightarrow P(\cdots, Z, \cdots)\]
- $\eta$ reduction [abstractions can be contracted]
  \[\lambda x. (P(x)) \Rightarrow P\]
- $\alpha$ reduction [renaming of variables]
Typed $\lambda$ calculus (Church 1940)

- The first form we introduced is called the $\beta, \eta$ long form, and the second more compact representation (which we use quite a bit below) is called the $\beta, \eta$ normal form. Here are some examples:

<table>
<thead>
<tr>
<th>$\beta, \eta$ normal form</th>
<th>$\beta, \eta$ long form</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>$\lambda x. \text{run}(x)$</td>
</tr>
<tr>
<td>every$^2$(kid, run)</td>
<td>every$^2((\lambda x. \text{kid}(x)), (\lambda x. \text{run}(x)))$</td>
</tr>
<tr>
<td>yesterday(run)</td>
<td>$\lambda y. \text{yesterday}(\lambda x. \text{run}(x))(y)$</td>
</tr>
</tbody>
</table>
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})) [honest!].
A grammar fragment

- $S : \beta(\alpha) \rightarrow NP : \alpha$  $VP : \beta$
  $NP : \beta(\alpha) \rightarrow Det : \beta$  $N' : \alpha$
  $N' : \beta(\alpha) \rightarrow Adj : \beta$  $N' : \alpha$
  $N' : \beta(\alpha) \rightarrow N' : \alpha$  $PP : \beta$
  $N' : \beta \rightarrow N : \beta$
  $VP : \beta(\alpha) \rightarrow V : \beta$  $NP : \alpha$
  $VP : \beta(\gamma)(\alpha) \rightarrow V : \beta$  $NP : \alpha$  $NP : \gamma$
  $VP : \beta(\alpha) \rightarrow VP : \alpha$  $PP : \beta$
  $VP : \beta \rightarrow V : \beta$
  $PP : \beta(\alpha) \rightarrow P : \beta$  $NP : \alpha$
A grammar fragment

- Kathy, NP: $kathy_{\text{Ind}}$
- Fong, NP: $fong_{\text{Ind}}$
- Palo Alto, NP: $paloalto_{\text{Ind}}$
- car, N: $\text{car}_{\text{Ind}} \rightarrow \text{Bool}$
- overpriced, Adj: $\text{overpriced}_{\text{Ind} \rightarrow \text{Bool} \rightarrow \text{Ind} \rightarrow \text{Bool}}$
- outside, PP: $\text{outside}_{\text{Ind} \rightarrow \text{Bool} \rightarrow \text{Ind} \rightarrow \text{Bool}}$
- red, Adj: $\lambda P. (\lambda x. P(x) \land \text{red}'(x))$
- in, P: $\lambda y. \lambda P. \lambda x. (P(x) \land \text{in}'(y)(x))$
- the, Det: $\iota$
- a, Det: $\text{some}^2_{\text{Ind} \rightarrow \text{Bool} \rightarrow \text{Ind} \rightarrow \text{Bool} \rightarrow \text{Bool}}$
- runs, V: $\text{run}_{\text{Ind} \rightarrow \text{Bool}}$
- respects, V: $\text{respect}_{\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool}}$
- likes, V: $\text{like}_{\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool}}$
A grammar fragment

- $\text{in}'$ is $\text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool}$
- $\text{in} \triangleq \lambda y.\lambda P.\lambda x.(P(x) \land \text{in}'(y)(x))$ is $\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool})$
- $\text{red}'$ is $\text{Ind} \rightarrow \text{Bool}$
- $\text{red} \triangleq \lambda P. (\lambda x.(P(x) \land \text{red}'(x)))$ is $(\text{Ind} \rightarrow \text{Bool}) \rightarrow (\text{Ind} \rightarrow \text{Bool})$
Model theory –
A formalization of a “database”

Properties

Stanford University
Natural Language Processing
Curried multi-argument functions

\[
\llbracket \text{respect} \rrbracket = \llbracket \lambda y. \lambda x. \text{respect}(x, y) \rrbracket = \begin{bmatrix}
    f & \rightarrow & 0 \\
    f & \rightarrow & k & \rightarrow & 1 \\
    b & \rightarrow & 0 \\
    f & \rightarrow & 1 \\
    k & \rightarrow & 1 \\
    b & \rightarrow & 0 \\
    f & \rightarrow & 1 \\
    b & \rightarrow & k & \rightarrow & 0 \\
    b & \rightarrow & 0
\end{bmatrix}
\]

\[
\llbracket \lambda x. \lambda y. \text{respect}(y)(x)(b)(f) \rrbracket = 1
\]
Adjective and PP modification

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

  \[
  \text{Adj} : \lambda P. (\lambda x. P(x) \land \text{red}'(x)) \\
  \text{N' : } \lambda x. (\text{car}(x) \land \text{in}'(\text{paloalto})(x)) \\
  \text{red} \\
  \text{N' : car} \\
  \text{PP : } \lambda P. \lambda x. (P(x) \land \text{in}'(\text{paloalto})(x)) \\
  \text{N : car} \\
  \text{P : } \lambda y. \lambda P. \lambda x. (P(x) \land \text{in}'(y)(x)) \\
  \text{NP : paloalto} \\
  \text{in} \\
  \text{Palo Alto}
  \]

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

  \[
  \text{Adj} : \lambda P. (\lambda x. P(x) \land \text{red}'(x)) \\
  \text{N' : } \lambda x. (\text{car}(x) \land \text{red}'(x)) \\
  \text{PP : } \lambda P. \lambda x. (P(x) \land \text{in}'(\text{paloalto})(x)) \\
  \text{N : car} \\
  \text{P : } \lambda y. \lambda P. \lambda x. (P(x) \land \text{in}'(y)(x)) \\
  \text{NP : paloalto} \\
  \text{in} \\
  \text{Palo Alto}
  \]
Intersective adjectives

• Syntactic ambiguity is spurious: you get the same semantics either way
• Database evaluation is possible via a table join

Non-intersective adjectives

• For non-intersective adjectives get different semantics depending on what they modify
  • overpriced(in(paloalto)(house))
  • in(paloalto)(overpriced(house))
• But probably won’t be able to evaluate it on database!
Adding more complex NPs

NP: A man $\rightarrow \exists x.\text{man}(x)$
S: A man loves Mary
$\rightarrow * \text{love}(\exists x.\text{man}(x),\text{mary})$

• How to fix this?
A disappointment

Our first idea for NPs with determiner didn’t work out:

“A man” \(\rightarrow\) \(\exists z.\text{man}(z)\)

“A man loves Mary” \(\rightarrow\) * \(\text{love}(\exists z.\text{man}(z),\text{mary})\)

But what was the idea after all?
Nothing!
\(\exists z.\text{man}(z)\) just isn’t the meaning of “a man”.

If anything, it translates the complete sentence
“There is a man”

Let’s try again, systematically…
A solution for quantifiers

What we want is:

“A man loves Mary” \( \rightarrow \exists z (\text{man}(z) \land \text{love}(z, \text{mary})) \)

What we have is:

“man” \( \rightarrow \lambda y. \text{man}(y) \)

“loves Mary” \( \rightarrow \lambda x. \text{love}(x, \text{mary}) \)

How about: \( \exists z (\lambda y. \text{man}(y)(z) \land \lambda x. \text{love}(x, \text{mary})(z)) \)

Remember: We can use variables for any kind of term.

So next:

\( \lambda P (\lambda Q. \exists z (P(z) \land Q(z))) \) \( \langle \sim \) “A”
Why things get more complex

• When doing predicate logic did you wonder why:
  - *Kathy runs* is \( \text{run}(\text{kathy}) \)
  - *no kid runs* is \( \neg (\exists x)(\text{kid}(x) \land \text{run}(x)) \)

• 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*?
- $\llbracket \iota \rrbracket(P) = a$ if $(P(b) = 1$ 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*

• NP : ι(λx.car(x) ∧ in′(paloalto)(x) ∧ red′(x))

```
<table>
<thead>
<tr>
<th>Det : ι</th>
<th>N' : λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>red car in Palo Alto</td>
</tr>
</tbody>
</table>
```

• *the red car in Palo Alto*

```sql
select Cars.obj from Cars, Locations, Red where
Cars.obj = Locations.obj AND
Locations.place = 'paloalto' AND Cars.obj = Red.obj
having count(*) = 1
```
Generalized Quantifiers

• What then of every red car in Palo Alto?

• A generalized determiner is a relation between two properties, one contributed by the restriction from the N′, and one contributed by the predicate quantified over:

\[(\text{Ind} \to \text{Bool}) \to (\text{Ind} \to \text{Bool}) \to \text{Bool}\]

• Here are some determiners

\[\text{some}^2(kid)(run) \equiv \text{some}(\lambda x.\text{kid}(x) \land \text{run}(x))\]

\[\text{every}^2(kid)(run) \equiv \text{every}(\lambda x.\text{kid}(x) \to \text{run}(x))\]
Generalized Quantifiers

- Generalized determiners are implemented via the quantifiers:

\[ \text{every}(P) = 1 \text{ iff } (\forall x)P(x) = 1; \]

i.e., if \( P = \text{Dom}_{\text{Ind}} \)

\[ \text{some}(P) = 1 \text{ iff } (\exists x)P(x) = 1; \text{ i.e., if } P \neq \emptyset \]
Generalized Quantifiers

- Every student likes the red car
- $S : \text{every}^2(\text{student})(\text{like}(\lambda x.\text{car}(x) \land \text{red}'(x)))$

\begin{align*}
\text{NP} : \text{every}^2(\text{student}) & \quad \text{VP : like}(\lambda x.\text{car}(x) \land \text{red}'(x)) \\
\text{Det : every}^2 & \quad \text{N' : student} \\
\text{every} & \quad \text{student} \\
\text{V : like} & \quad \text{Det : } t \\
\text{likes} & \quad \text{N' : } \lambda x.\text{(car}(x) \land \text{red}'(x)) \\
\text{the} & \quad \text{Adj : } \lambda P.\text{(car}(x) \land \text{red}'(x)) \\
\text{red} & \quad \text{N : car} \\
\text{car} & 
\end{align*}
Questions with answers!

- A yes/no question (*Is Kathy running?*) will be something of type **Bool**, checked on database
- A content question (*Who likes Kathy?*) will be an *open proposition*, that is something semantically of the type *property* (**Ind** → **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

• \( S' : \beta(\alpha) \rightarrow NP[wh] : \beta \)  \( \text{Aux} \)  \( S : \alpha \)
  \( S' : \alpha \rightarrow \text{Aux} \)  \( S : \alpha \)
  \( NP/NP_Z : z \rightarrow e \)
  \( S : \lambda z.F(...z...) \rightarrow S/\text{NP}_Z : F(...z...) \)
Syntax/semantics for questions

- **who**, NP[wh] : $\lambda U.\lambda x.U(x) \land \text{human}(x)$
  - **what**, NP[wh] : $\lambda U.U$
  - **which**, Det[wh] : $\lambda P.\lambda V.\lambda x.P(x) \land V(x)$
  - **how many**, Det[wh] : $\lambda P.\lambda V.|\lambda x.P(x) \land V(x)|$

- Where $|\cdot|$ is the operation that returns the cardinality of a set (count).
**Question examples**

- \( S' : \lambda z. \text{like}(z)(\text{kathy}) \)

  \( \text{NP}[\text{wh}] : \lambda U.U \quad \text{Aux} \quad S : \lambda z. \text{like}(z)(\text{kathy}) \)

  \( S/\text{NP}_z : \text{like}(z)(\text{kathy}) \)

  \( \text{NP} : \text{kathy} \quad \text{VP}/\text{NP}_z : \text{like}(z) \)

  \( \text{Kathy} \quad \text{V} : \text{like} \quad \text{NP}/\text{NP}_z : z \)

  \( \text{like} \quad \text{e} \)

- select liked from Likes where Likes.liker=’Kathy’
Question examples

- \( S' : \lambda x. \text{like}(x)(\text{kathy}) \land \text{human}(x) \)

  \[
  \begin{align*}
  \text{NP[wh]} : & \lambda U. \lambda x. U(x) \land \text{human}(x) \\
  \text{Aux} : & S : \lambda z. \text{like}(z)(\text{kathy}) \\
  \text{S/NP}_z : & \text{like}(z)(\text{kathy}) \\
  \text{NP :} & \text{kathy} \\
  \text{VP/NP}_z : & \text{like}(z) \\
  \text{Kathy} : & \text{like} \\
  \text{V :} & \text{like} \\
  \text{NP/NP}_z : & z \\
  \text{like} : & e
  \end{align*}
  \]

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

- \( S' : \lambda x.\text{car}(x) \land \text{like}(x)(\text{kathy}) \)

- \( \text{NP[wh]} : \lambda V.\lambda x.\text{car}(x) \land V(x) \)

- \( \text{Aux} \)

- \( \text{S} : \lambda z.\text{like}(z)(\text{kathy}) \)

- \( \text{S/NP} : \text{like}(z)(\text{kathy}) \)

- \( \text{Det} : \lambda P.\lambda V.\lambda x.P(x) \land V(x) \)

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

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

- \( \text{did} \)

- \( \text{NP} : \text{car} \)

- \( \text{VP/NP}_z : \text{like}(z) \)

- \( \text{NP} : \text{kathy} \)

- \( \text{V} : \text{like} \)

- \( \text{NP/NP}_z : z \)

- \( \text{like} \)

- \( \text{e} \)

- \( \text{select liked from Cars,Likes where Cars.obj=Likes.liked AND Likes.liker='Kathy'} \)
Question examples

- $S' : \lambda x.\text{car}(x) \land \text{every}^2(\text{student})(\text{like}(x))$

  - $\text{NP[wh]} : \lambda V.\lambda x.\text{car}(x) \land V(x)$
    - $\text{Det} : \lambda P.\lambda V.\lambda x. P(x) \land V(x)$
      - $\text{N'} : \text{car}$
        - $\text{Which}$
        - $\text{N} : \text{car}$
          - $\text{cars}$

  - $\text{Aux}$
    - $\lambda z.\text{every}^2(\text{student})(\text{like}(z))$
      - $\text{VP}$
        - $\text{NP}_z : \text{like}(z)$
          - $\text{V} : \text{like}$
            - $\text{NP}$
              - $\text{NP}_z : z$
                - $\text{e}$
Question examples

- **How many red cars in Palo Alto does Kathy like?**
  
  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

- **Did Kathy see the red car in Palo Alto?**
  
  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?
Did Kathy see the red car in Palo Alto?

S' : see(ι(λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x)))(kathy)

Aux  S : see(ι(λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x)))(kathy)

Did  NP : kathy  VP : see(ι(λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x)))

Kathy  V : see  NP : ι(λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x))

see  Det : ι  N' : λx.car(x) ∧ in'(paloalto)(x) ∧ red'(x)

the  Adj : λP.(λx.P(x) ∧ red'(x))  N' : λx.(car)

red  N : car  P : λx.
How could we learn such representations?

- After disengagement for many years, there has started to be very interesting work in this area:
How could we learn such representations?

- General approach (ZC05): Start with initial lexicon, category templates, and paired sentences and meanings:
  
  What states border Texas?
  
  $$\lambda x. \text{state}(x) \land \text{borders}(x, \text{texas})$$

- Learn lexical syntax/semantics for other words and learn to parse to logical form (parse structure is hidden).

- They successfully do iterative refinement of a lexicon and maxent parser
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 – Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, Benjamin Taskar. 2007. Probabilistic Relational Models. In An Introduction to Statistical Relational Learning. MIT Press.
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.