An Introduction to Formal Computational Semantics

CS224N/Ling 280

Christopher Manning

May 23, 2000; revised 2008
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

$S : \beta(\alpha) \rightarrow \text{NP} : \alpha \quad \text{VP} : \beta$

$\text{VP} : \beta(\alpha) \rightarrow \text{V} : \beta \quad \text{NP} : \alpha$

$\text{VP} : \beta \rightarrow \text{V} : \beta$
A first example

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

  NP : kathy
  
  VP : $\lambda x. \text{respect}(x, \text{fong})$

- $[\text{VP respects Fong}] : [\lambda y. \lambda x. \text{respect}(x, y)](\text{fong})$
  $= \lambda x. \text{respect}(x, \text{fong})$ [β red.]

- $[S \text{ Kathy respects Fong}] : [\lambda x. \text{respect}(x, \text{fong})](\text{kathy})$
  $= \text{respect(\text{kathy}, \text{fong})}$
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)
  ```
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!)
- \( \text{Bool} \) truth values (0 and 1)
  \( \text{Ind} \) individuals
  \( \text{Ind} \rightarrow \text{Bool} \) properties
  \( \text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool} \) binary relations
- kathy and fong are \( \text{Ind} \)
  run is \( \text{Ind} \rightarrow \text{Bool} \)
  respect is \( \text{Ind} \rightarrow \text{Ind} \rightarrow \text{Bool} \)
- Types are interpreted right associatively.
  respect is \( \text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Bool}) \)
- We convert a several argument function into embedded unary functions. Referred to as \textit{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>$every^2(\text{kid, run})$</td>
<td>$every^2((\lambda x.\text{kid}(x)), (\lambda x.\text{run}(x))$</td>
</tr>
<tr>
<td>yesterday(\text{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 (\texttt{Ind} → \texttt{Bool})

• noun phrases are \texttt{Ind} – though they are commonly type-raised to (\texttt{Ind} → \texttt{Bool}) → \texttt{Bool}

• adjectives are (\texttt{Ind} → \texttt{Bool}) → (\texttt{Ind} → \texttt{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 ((\texttt{Ind} → \texttt{Bool}) → (\texttt{Ind} → \texttt{Bool})) → ((\texttt{Ind} → \texttt{Bool}) → (\texttt{Ind} → \texttt{Bool})) [honest!].
A grammar fragment

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

- **Kathy**, NP: $\text{kathy}_{\text{Ind}}$
- **Fong**, NP: $\text{fong}_{\text{Ind}}$
- **Palo Alto**, NP: $\text{paloalto}_{\text{Ind}}$
- **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

- \texttt{in}' is \texttt{Ind} \to \texttt{Ind} \to \texttt{Bool}
- \texttt{in} \overset{\text{def}}{=} \lambda y.\lambda P.\lambda x. (P(x) \land \texttt{in}'(y)(x)) \text{ is } \texttt{Ind} \to (\texttt{Ind} \to \texttt{Bool}) \to (\texttt{Ind} \to \texttt{Bool})
- \texttt{red}' is \texttt{Ind} \to \texttt{Bool}
- \texttt{red} \overset{\text{def}}{=} \lambda P.(\lambda x. (P(x) \land \texttt{red}'(x))) \text{ is } (\texttt{Ind} \to \texttt{Bool}) \to (\texttt{Ind} \to \texttt{Bool})
Model theory –
A formalization of a “database”
Curried multi-argument functions

\[ \text{\textbf{尊敬}} = \text{\textbf{λ}y.λx.尊敬(}x, y)\text{\textbf{)}} \]

\[ = \begin{cases} f & \rightarrow 0 \\ f & \rightarrow k & \rightarrow 1 \\ b & \rightarrow 0 \\ f & \rightarrow 1 \\ k & \rightarrow k & \rightarrow 1 \\ b & \rightarrow 0 \\ f & \rightarrow 1 \\ b & \rightarrow k & \rightarrow 0 \\ b & \rightarrow 0 \end{cases} \]

\[ \text{\textbf{λ}x.λy.尊敬(y)(}x(}b(}f)\text{\textbf{)}} \]

= 1
 Quiz question

- Which individuals are the red things in Palo Alto?
- Who respects kathy (k)?
Adjective and PP modification

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

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

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

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

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

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

N : \text{car}

P : \( \lambda y. \lambda P. \lambda x. (P(x) \land \text{in}'(y)(x)) \)

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

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

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

NP : \text{paloalto}

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

NP : \text{paloalto}

14
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 ~> \( \exists x. \text{man}(x) \)
S: A man loves Mary

~> * love(\( \exists x. \text{man}(x) \), 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:

\(\lambda P (\lambda Q. \exists z (P(z) \land Q(z))) \leftarrow \text{“A”}\)
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*

  ```sql
  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**

  \[
  \text{NP} : \iota(\lambda x. \text{car}(x) \land \text{in'}(\text{paloalto})(x) \land \text{red'}(x))
  \]

  \[
  \begin{array}{c}
  \text{Det} : \iota \\
  \text{N' : } \lambda x. \text{car}(x) \land \text{in'}(\text{paloalto})(x) \land \text{red'}(x) \\
  \end{array}
  \]

- **the 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
  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

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

  \[
  \text{every}^2(\text{kid})(\text{run}) \equiv \text{every}(\lambda x. \text{kid}(x) \to \text{run}(x))
  \]
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
• $S: \text{every}^2(\text{student})\ (\text{like}(\iota(\lambda x.\text{car}(x) \land \text{red}'(x))))$

\[
\begin{align*}
\text{Det: } & \text{every}^2 & \text{N': } & \text{student} \\
\text{NP: } & \text{every}^2(\text{student}) & \text{VP: } & \text{like}(\iota(\lambda x.\text{car}(x) \land \text{red}'(x))) \\
\text{V: } & \text{like} & \text{NP: } & \iota(\lambda x.\text{car}(x) \land \text{red}'(x)) \\
\text{Det: } & \iota & \text{N': } & \lambda x.(\text{car}(x) \land \text{red}'(x)) \\
\text{Adj: } & \lambda P.(\lambda x.P(x) \land \text{red}'(x)) & \text{N': } & \text{car} \\
\text{N: } & \text{car} & &
\end{align*}
\]
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: λP.P(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

\[ \text{Ind} \xrightarrow{l} \text{Ind} \rightarrow \text{Bool} \]

\[ R(x) = \lambda P. P(x) \]
\[ \text{some}^2(P) = \lambda Q. (Q \cap P) \neq \emptyset \]
\[ Q(x) = \lambda y. x = y \]

- In this diagram, \( R \) is exactly this basic type-raising function for individuals.
Value raising raises a function that produces an individual to one that produces a quantifier. If \( \alpha : \sigma \to \text{Ind} \) then \( \lambda x.\lambda P.P(\alpha(x)) : \sigma \to (\text{Ind} \to \text{Bool}) \to \text{Bool} \)

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

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 \( \alpha : \sigma \to ((\text{Ind} \to \text{Bool}) \to \text{Bool}) \to \tau \to \text{Bool} \) then \( \lambda x_1.\lambda x_2.\lambda x_3.\alpha(x_1)(\lambda P.P(x_2))(x_3) : \sigma \to \text{Ind} \to \tau \to \text{Bool} \)
Every student runs

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

```
Det : every^2                  NP : every^2(\text{student})
  |                          |VP : \lambda Q. Q(\lambda x.\text{run}(x))
|   every                  |VP : \text{run}
|   N' : \text{student}   |V : \text{run}
|   N : \text{student}    |    runs
|student
```
Some kid broke every toy

- $S : \text{every}^2(\text{toy})(\lambda y_0.\text{some}^2(\text{kid})(\lambda x_5.\text{break}(y_0)(x_5)))$

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

- $\text{VP} : \lambda S'. \text{every}^2(\text{toy})(\lambda y_0.S'(\lambda x_5.\text{break}(y_0)(x_5)))$

- $\text{Det} : \text{some}^2 \quad \text{N'} : \text{kid}$

- $\text{V} : \lambda O.\lambda S'.O(\lambda y_0.S'(\lambda x_5.\text{break}(y_0)(x_5)))$

- $\text{NP} : \text{every}^2(\text{toy})$

- $\text{Det} : \text{every}^2 \quad \text{N'} : \text{toy}$

- $\text{N} : \text{toy}$

- $\text{V} : \lambda y_0.\lambda s.\text{break}(y_0)(x_5)$

- $\text{V} : \lambda y_0.\lambda S.'.O(\lambda y_0.S'(\lambda x_5.\text{break}(y_0)(x_5)))$

- $\text{V} : \lambda x_0.\lambda S.O(\lambda x_5.\text{break}(x_0)(x_5))$

- $\text{V} : \lambda y.\lambda x.\text{break}(y)(x)$

- $\text{broke}$

- $\text{every}$

- $\text{toy}$
Some kid broke every toy

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

- $S' : \beta(\alpha) \rightarrow NP[wh]:\beta$  Aux  $S : \alpha$
- $S' : \alpha \rightarrow Aux  S : \alpha$
- $NP/NP_Z : z \rightarrow e$
- $S : \lambda z. F(...z...) \rightarrow S/NP_Z : F(...z...)$
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

- \( S' : \lambda z.\text{like}(z) (\text{kathy}) \)
  
  \[ 
  \begin{align*}
  \text{NP}[\text{wh}] : \lambda U.U & \quad \text{Aux} \quad S : \lambda z.\text{like}(z) (\text{kathy}) \\
  \text{What} & \quad \text{does} \\
  & \quad \text{S/NP}_z : \text{like}(z) (\text{kathy}) \\
  & \quad \text{NP} : \text{kathy} \quad \text{VP/NP}_z : \text{like}(z) \\
  & \quad \text{Kathy} \quad \text{V} : \text{like} \quad \text{NP}/\text{NP}_z : z \\
  & \quad \text{like} \quad \text{NP} \quad \text{e} 
  \end{align*} \]

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

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

  NP[$wh$] : $\lambda U.\lambda x.U(x) \land \text{human}(x)$

  Aux

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

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

  NP : $\text{kathy}$

  VP/NP$_Z$ : $\text{like}(z)$

  Kathy

  V : $\text{like}$

  NP/NP$_Z$ : $z$

  like

  e

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

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

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

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

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

- \( \text{Aux} \)

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

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

- \( \text{Which} \)

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

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

- \( \text{Kathy} \)

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

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

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

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

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

  Which

  N : car

  cars

  $S : \lambda z.\text{every}^2(\text{student})(\text{like}(z))$

  S/NP : $\text{every}^2(\text{student})(\text{like}(z))$

• $???

  NP : $\text{every}^2(\text{student})$

  VP

  NP$_z$ : $\text{like}(z)$

  Det : $\text{every}^2$  
  N' : student

  every

  student

  V : $\text{like}$

  NP

  NP$_z$ : $z$

  like

  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?

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