LING 364: Introduction to Formal Semantics

Lecture 19
March 20th
Administrivia

• Handout: Chapter 6
  – Quantifiers
  – hard topic
  – we’ll start on it today

• Read it for next Tuesday
  – Short Quiz 5
Administrivia

• We’ll review Homework 4 next time
  – a bit behind on grading...
Leftover from Last Lecture

• Example:
  – (29) Only John loves his mother
  – (29’) Only John doesn’t love his mother

• World 1 for (29) (=31):
  – loves(john,mother(john)).
  – also, no other facts in the database that would satisfy the query
  – ?- loves(X,mother(john)), \+ X=john.

• World 2 for (29) (=32):
  – loves(john,mother(john)).
  – also no other facts in the database that would satisfy the query
  – ?- loves(X,mother(X)), \+ X=john.

• Both Worlds are possible since (29) is ambiguous
• Which one is preferred?
Leftover from Last Lecture

• **Example:**
  - (29) Only John loves his mother
  - (29') Only John *doesn’t* love his mother

• **World 3 for (29’):**
  - `loves_not(john,mother(john)).`
  - also, no other facts in the database that would satisfy the query
  - `?- loves_not(X,mother(john)), \+ X=john.`

• **World 4 for (29’):**
  - `loves_not(john,mother(john)).`
  - also no other facts in the database that would satisfy the query
  - `?- loves_not(X,mother(X)), \+ X=john.`

• Both Worlds are possible since *(presumably)* (29’) is also ambiguous like (29)

• Which one is preferred?
Today’s Topic

• Chapter 6: Quantifiers
Quantifiers

• Not all noun phrases (NPs) are (by nature) directly referential like names
• **Quantifiers**: 
  – “*something to do with indicating the quantity of something*”
• **Examples**: 
  – every child
  – nobody
  – two dogs
  – several animals
  – most people

  – nobody has seen a unicorn
  – could simply means *something like (Prolog-style)*:
  – `?- findall(X,(person(X), seen(X,Y), unicorn(Y)),Set),length(Set,0).`
Quantifiers

• Recall: compositionality idea:
  – elements of a sentence combine in piecewise fashion to form an overall (propositional) meaning for the sentence

• Example:
  – (4) Every baby cried
    – **Word**                   **Meaning**
    – cried                     cried(X).
    – baby                      baby(X).
    – **every**                  ?
    – every baby cried          proposition (True/False)
    –                            that can be evaluated in a given world
Quantifiers

- **Scenario (Possible World):**
  - suppose there are three babies...
    - baby(noah).
    - baby(merrill).
    - baby(dani).
  - all three cried
    - cried(noah).
    - cried(merrill).
    - cried(dani).
  - only Dani jumped
    - jumped(dani).
  - Noah and Dani swam
    - swam(noah).
    - swam(dani).

<table>
<thead>
<tr>
<th></th>
<th>every baby</th>
<th>exactly one baby</th>
<th>most babies</th>
</tr>
</thead>
<tbody>
<tr>
<td>cried</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>jumped</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>swam</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

- think of quantifiers as “properties-of-properties”
- every_baby(P) is a proposition
- P: property
- every_baby(P) true for P=cried
- every_baby(P) false for P=jumped and P=swam
Quantifiers

- **think of quantifiers as “properties-of-properties”**
  - every_baby(P) **true** for P=cried
  - every_baby(P) **false** for P=jumped and P=swam

- **Generalized Quantifiers** *(scary jargon alert!)*
  - the idea that quantified NPs represent sets of sets
  - *this idea is not as weird as it sounds*
  - we know
    - every_baby(P) is true for certain properties
  - view
    - every_baby(P) = set of all properties P for which this is true
  - in our scenario
    - every_baby(P) = {cried}
  - we know *cried* can also be view as a set itself
    - cried = set of individuals who cried
  - in our scenario
    - cried = {noah, merrill, dani}
Quantifiers

• **how do we define the expression every_baby(P)?**

• *(Montague-style)*

  every_baby(P) is shorthand for
  
  – for all individuals X, baby(X) \(\rightarrow\) P(X)
  
  \(\rightarrow:\) *if-then (implication: logic symbol)*

• written another way *(lambda calculus-style):*

  – \(\lambda P. [\forall X. [\text{baby}(X) \rightarrow P(X)]]\)
  
  \(\forall:\) *for all (universal quantifier: logic symbol)*

• **Example:**

  – every baby walks
    
    – for all individuals X, baby(X) \(\rightarrow\) walks(X)
    
    – more formally
    
    – \([_{NP} \text{every baby}] [_{VP} \text{walks}]\)
      
      – \(\lambda P. [\forall X. [\text{baby}(X) \rightarrow P(X)]](\text{walks})\)
      
      – \(\forall X. [\text{baby}(X) \rightarrow \text{walks}(X)]\)
Quantifiers

- **how do we define this Prolog-style?**

- **Example:**
  - every baby walks
  - $\left[_{NP} \text{every baby} \right] \left[_{VP} \text{walks} \right]
    - $\lambda P. \left( \forall X (\text{baby}(X) \rightarrow P(X)) \right)$(walks)
    - $\forall X (\text{baby}(X) \rightarrow \text{walks}(X))$

- **Possible World (Prolog database):**
  - `:- dynamic baby/1.`
  - `baby(a). baby(b).`
  - `walks(a). walks(b). walks(c).`
  - `individual(a). individual(b). individual(c).`

- **What kind of query would you write?**

- **One Possible Query (every means there are no exceptions):**
  - `?- \+ (\text{baby}(X), \+ \text{walks}(X)).` (NOTE: need a space between \+ and ( here)
  - Yes (TRUE)

  - `?- \text{baby}(X), \+ \text{walks}(X).`
  - No

- `?- assert(baby(d)).`
  - `?- \text{baby}(X), \+ \text{walks}(X).`
  - `X = d ;`
  - Yes

using idea that $\forall X P(X)$ is the same as $\neg \exists X \neg P(X)$
$\exists$ = “there exists” (quantifier)
(implicitly: all Prolog variables are existentially quantified variables)
Aside: Truth Tables

- **logic of implication**
- P -> Q = \((truth\ value)\)
- P -> Q =
- T T T
- F T T
- F F T
- T F F
- i.e. if P is true, Q must be true in order for P->Q to be true
- if P is false, doesn’t matter what Q is, P->Q is true
- conventionally written as:

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P -&gt; Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Hence, P->Q is equivalent to \(\neg P \lor Q\)

PvQ=F only when both P and Q are F

\(\neg P \lor Q=F\) only when P=T and Q=F

P->Q=F only when P=T and Q=F

\(\neg P\lor Q=F\) only when both P and Q are F
Aside: Truth Tables

- De Morgan’s Rule
- \( \neg(P \lor Q) = \neg P \land \neg Q \)

<table>
<thead>
<tr>
<th>( \neg P )</th>
<th>( \land )</th>
<th>( \neg Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>F</td>
<td>FT</td>
</tr>
<tr>
<td>TF</td>
<td>F</td>
<td>FT</td>
</tr>
<tr>
<td>TF</td>
<td>T</td>
<td>TF</td>
</tr>
<tr>
<td>FT</td>
<td>F</td>
<td>FT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>v</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\( \neg(P \lor Q) \) = T only when both P and Q are F

\( \neg P \land \neg Q \) = T only when both P and Q are F

Hence, \( \neg(P \lor Q) \) is equivalent to \( \neg P \land \neg Q \)
Conversion into Prolog

Note: 

\[ \neg (\text{baby}(X), \neg \text{walks}(X)) \]

is Prolog for \[ \forall X (\text{baby}(X) \rightarrow \text{walks}(X)) \]

Steps:

1. \[ \forall X (\text{baby}(X) \rightarrow \text{walks}(X)) \]
2. \[ \forall X (\neg \text{baby}(X) \lor \text{walks}(X)) \]
   - (since \( P \rightarrow Q = \neg P \lor Q \), see truth tables from two slides ago)
3. \[ \neg \exists X \neg (\neg \text{baby}(X) \lor \text{walks}(X)) \]
   - (since \( \forall X P(X) = \neg \exists X \neg P(X) \), no exception idea from 3 slides ago)
4. \[ \neg \exists X (\text{baby}(X) \land \neg \text{walks}(X)) \]
   - (by De Morgan's rule, see truth table from last slide)
5. \[ \neg (\text{baby}(X) \land \neg \text{walks}(X)) \]
   - (can drop \( \exists X \) since all Prolog variables are basically existentially quantified variables)
6. \[ \neg (\text{baby}(X) \land \neg \text{walks}(X)) \]
   - (\( \neg \) = Prolog negation symbol)
7. \[ \neg (\text{baby}(X), \neg \text{walks}(X)) \]
   - (\( \neg \) = Prolog conjunction symbol)
Quantifiers

- **how do we define this Prolog-style?**
- **Example:**
  - every baby walks
  - \([_{NP} \text{every baby}][_{VP} \text{walks}]\)
    - \(\forall P. [\forall X. [\text{baby}(X) \rightarrow P(X)]](\text{walks})\)
    - \(\forall X. [\text{baby}(X) \rightarrow \text{walks}(X)]\)
- **Another Possible World (Prolog database):**
  - \(- \text{dynamic baby}/1.\)
  - \(- \text{dynamic walks}/1.\)
  - \(^\% \text{no facts} \quad (^\% = \text{comment})\)
- **Does \(?- \lnot (\text{baby}(X), \lnot \text{walks}(X)). \text{still work?}\)**
  - **Yes because**
    - \(?- \text{baby}(X), \lnot \text{walks}(X).\)
    - \(\text{No}\)
  - cannot be satisfied
Quantifiers

• how do we define the expression every_baby(P)?
  • (Montague-style)
    every_baby(P) is shorthand for
    – \( \lambda P. [\forall x. \text{baby}(x) \rightarrow P(x)] \)

• (Barwise & Cooper-style)
  • think directly in terms of sets
  • leads to another way of expressing the Prolog query

• Example: every baby walks
  • \{X: \text{baby}(X)\} set of all X such that baby(X) is true
  • \{X: \text{walks}(X)\} set of all X such that walks(X) is true

• Subset relation \( (\subseteq) \)
  • \{X: \text{baby}(X)\} \subseteq \{X: \text{walks}(X)\} the “baby” set must be a subset of the “walks” set
Quantifiers

- (Barwise & Cooper-style)
  - think directly in terms of sets
  - *leads to another way of expressing the Prolog query*

- **Example**: every baby walks
  - \( \{X: \text{baby}(X)\} \subseteq \{X: \text{walks}(X)\} \)  
    *the “baby” set must be a subset of the “walks” set*

- **How to express this as a Prolog query?**
  - **Queries**:
    - `?- findall(X,baby(X),L1).`  
      _L1 is the set of all babies in the database_
    - `?- findall(X,walks(X),L2).`  
      _L2 is the set of all individuals who walk_

| Need a Prolog definition of the subset relation. This one, for example: |
| subset([],_). |
| subset([X|L1],L2) :- member(X,L2), subset(L1,L2). |
| member(X,[X|_]). |
| member(X,[_|L]) :- member(X,L). |
Quantifiers

- **Example:** every baby walks
  - \(\{X: \text{baby}(X)\} \subseteq \{X: \text{walks}(X)\}\) the “baby” set must be a subset of the “walks” set

- **Assume the following definitions are part of the database:**
  
  \[
  \begin{align*}
  \text{subset}([],_) & . \\
  \text{subset}([X|_],L) & :- \text{member}(X,L). \\
  \text{member}(X,[X|_]) & . \\
  \text{member}(X,[_|L]) & :- \text{member}(X,L).
  \end{align*}
  \]

- **Prolog Query:**
  - `?- findall(X,baby(X),L1), findall(X,walks(X),L2), subset(L1,L2).`

- **True for world:**
  - baby(a).
  - baby(b).
  - walks(a).
  - walks(b).
  - walks(c).
  
  \[
  \begin{align*}
  L1 & = [a,b] \\
  L2 & = [a,b,c]
  \end{align*}
  \]
  
  `- subset(L1,L2) is true`

- **False for world:**
  - baby(a).
  - baby(b).
  - baby(d).
  - walks(a).
  - walks(b).
  - walks(c).
  
  \[
  \begin{align*}
  L1 & = [a,b,d] \\
  L2 & = [a,b,c]
  \end{align*}
  \]
  
  `- subset(L1,L2) is false`