LING 364: Introduction to Formal Semantics

Lecture 10
February 14th
Administrivia

• Reminder
  – Homework 2 due tonight

  • we did Exercises 1 through 3 in the lab class last Thursday

  • need more help?
  • see me after class today...
Administrivia

• **Thursday**
  – (3:30pm – 4:45pm)
    • Computer Lab Class
    • meet in Social Sciences 224 instead of here
Last Time

• **Grammar Rule Recursion**

• Recursion:
  – A phrase may contain embedded inside another instance of the same phrase

• Examples:
  – sentence with a *relative clause*
    • $[\text{Sbar} \; [\text{S} \; \text{I saw} \; [\text{NP} \; \text{the man} \; [\text{Sbar} \; \text{who} \; [\text{S} \; \text{attacked me}]]]]]]$
  – *possessive NPs*
    • $[[\text{NP} \; [\text{NP} \; [\text{NP} \; \text{John’s mother’s cat}]]]]$
Last Time

- **Grammar Rule Recursion**

- *(Fixed) Prolog Computation Rule:* 
  - always pick the *first-mentioned* matching grammar rule to try each time we expand a non-terminal

- **General Rule for writing recursive rules:** 
  - put recursive case **last**
  - i.e. *place non-recursive rules for a non-terminal ahead of the recursive ones*

- **DCG rules for Possessive NPs:**
  - `np --> np, ["'s"], n.`
  - `n --> [mother].`
  - `n --> [cat].`
  - `np --> [john].`
Last Time

• Chapter 3: More about Predicates
  • Lambda Calculus vs. Prolog notation
    – easy to understand as just “syntactic sugar”
      • i.e. just an equivalent way of expressing what we’ve been using Prolog for
    – every logic variable, e.g. \(X\), must be “quantified” using lambda, e.g. \(\lambda x.\)
    – result is a slightly more complicated-looking notation
  • Example:
    – Phrase                           Lambda Calculus                        Prolog notation
      – barks                             \(\lambda x. x \text{ barks}\)            barks(X).
      – Shelby barks                       \([\lambda x. \text{ barks}](\text{Shelby})\)   barks(X), X = shelby.
  • Example (Quiz 3) transitive predicate:
    – Phrase                           Lambda Calculus                        Prolog notation
      – likes                             \(\lambda y.[\lambda x. x \text{ likes } y]\)   likes(X,Y).
      – likes Mary                        \([\lambda y.[\lambda x. x \text{ likes } y]](\text{Mary})\) likes(X,Y), Y = mary.
Today’s Topic

• “The Lambda Calculus Lecture”
  – Getting comfortable with Lambda Calculus
    • see it as another way of stating what we have been doing already using Prolog notation
  – do lots of examples
More on the Lambda Calculus

- **Lambda Calculus** vs. Prolog notation
- Example (Quiz 3) *transitive predicate*:
  - **Phrase** | **Lambda Calculus** | **Prolog notation**
  - *likes* | λy.[λx.x likes y] | likes(X,Y).
  - *likes Mary* | ([λy.[λx.x likes y]](Mary) | likes(X,Y), Y = mary.
  - | λx.x likes Mary | likes(X,mary).
  - *John likes Mary* | ([λx.x likes Mary])(John) | likes(X,mary), X = john.
  - | John likes Mary | likes(john,mary).
More on the Lambda Calculus

- How to do variable substitution
  - Official Name: Beta (β)-reduction

  - Example Expression
  - \textit{likes} \quad [\lambda y.[\lambda x. x \text{ likes } y]]
  - \textit{likes Mary} \quad [\lambda y.[\lambda x. x \text{ likes } y]](\text{Mary})

  - means (basically):
  - (1) delete the outer layer, i.e. remove $[\lambda y. \Box](\text{Mary})$ part, and
  - (2) keep the $\Box$ part, and
  - (3) replace all occurrences of the deleted lambda variable $y$ in $\Box$ with Mary

\[
[\lambda y.[\lambda x. x \text{ likes } y]](\text{Mary})
\]

\[
[\lambda x. x \text{ likes } y] \quad [\lambda y. \Box](\text{Mary})
\]

\[
[\lambda x. x \text{ likes } \text{Mary}]
\]
More on the Lambda Calculus

**Note:**

nesting order of \( \lambda y \) and \( \lambda x \) matters

**why:**

\[ \lambda y. [\lambda x. x \text{ likes } y] \]
\[ \lambda x. [\lambda y. x \text{ likes } y] \]

**here:** lambda expression quantifier for the object must be outside because of phrase structure hierarchy

**Example:**

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Lambda Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>likes</td>
<td>( \lambda y. [\lambda x. x \text{ likes } y] )</td>
</tr>
<tr>
<td>likes Mary</td>
<td>( <a href="%5Ctext%7BMary%7D">\lambda y. [\lambda x. x \text{ likes } y]</a> )</td>
</tr>
<tr>
<td>John likes Mary</td>
<td>( <a href="%5Ctext%7BMary%7D">\lambda x. x \text{ likes } y</a> )</td>
</tr>
</tbody>
</table>

Prolog notation

Lambda Calculus
More on the Lambda Calculus

• 3.3 Relative Clauses
  – (7) Hannibal is [who Shelby saw]

• semantics of relative clause [who Shelby saw]:
  – who Shelby saw is a bit like a sentence (proposition)
    • who₁ Shelby saw e₁ wh-movement of who₁ leaving a trace e₁
    • Shelby saw who underlying structure

• Prolog style:
  • saw(shelby,who).
  • saw(shelby,X).
    (using a logic variable for who)

• lambda calculus style:
  • λx.Shelby saw x (straight translation from Prolog)
More on the Lambda Calculus

- We’re going to compare:
  - (7) Hannibal is [who Shelby saw]
  - (7’) Hannibal is happy
- Consider the semantics of (7’)

  cf. Homework 2
  John is a student  student(john).
  John is a baseball fan  baseball_fan(john).

- In the lambda calculus, the semantics of copula *be* is the identity function, e.g. λy.y
- Example Derivation:
  - Phrase         Lambda Calculus
    - is             λy.y
    - happy          λx.x happy
    - is happy       [λy.y](λx.x happy)
    - λx.x happy

basically the same derivation as...
Phrase         Lambda Calculus
barks          λx.x barks
Shelby barks  [λx.x barks](Shelby)
Shelby barks
More on the Lambda Calculus

• Back to comparing:
  – (7) Hannibal is [who Shelby saw]
  – (7’) Hannibal is happy

• Semantics (Prolog-style):
  – (7) Hannibal is [saw(shelby,X)]
  – (7’) Hannibal is [happy(X)]

• Semantics (Lambda calculus):
  – (7) Hannibal is [λx.Shelby saw x]
  – (7’) Hannibal is [λx.x happy]

• Notice the similarity between (7) and (7’) wrt meaning:
  – both highlighted parts are single variable λx expressions
  – (unsaturated for subject)
  – we can say they are of the “same type”
  – This means we can use the same identity function λy.y for the copula in either case

(Simplified Derivation)
Points to remember:
Phrase | Lambda calculus
who    | λx
e      | x
More on the Lambda Calculus

- We could do topicalization in the same way as for relative clauses

3.4 Topicalization
- (9) Shelby, Mary saw
- (10) Shelby is who₁ Mary saw e₁
- (10’) Shelby is [λx.Mary saw x]
- (10”) Mary saw Shelby