LING 364: Introduction to Formal Semantics

Lecture 7

February 2nd
Administrivia

• today
  – (3:30pm – 4:40pm)
    • lecture here in Comm 214
  – (4:45pm – 5:45pm) (EXTRA)
    • lab practice in Social Sciences Lab 224

• also next week...
  – see schedule in Lecture 6 slides
Last Time

• **Compositionality**: meaning of a sentence is composed from the meaning of its subparts

• **example**:
  – given “John likes Mary” corresponds to likes(john, mary).
  – meaning fragments are
    • **word or phrase** meaning
      • John john
      • likes Mary likes(X, mary).
      • likes likes(X, Y).
      • Mary mary
    – each word here has a contribution to make to the meaning of the complete sentence
  – cf. it is raining (pleonastic “it”/ambient “it”)

\[ \text{likes(john, mary)} \]
\[ \text{john} \quad \text{likes(X, mary)} \]
\[ \text{John} \quad \text{likes(X, Y)} \quad \text{mary} \]
\[ \text{likes} \quad \text{Mary} \]

\[ \text{likes(john, mary)} \]
\[ \text{john} \quad \text{likes(X, mary)} \]
\[ \text{John} \quad \text{likes(X, Y)} \quad \text{mary} \]
\[ \text{likes} \quad \text{Mary} \]
Last Time

• Language violates compositionality in the case of idioms
• example:
  – John kicked the bucket
  – **literal meaning:**
    • word meaning
    • john john
    • kick kick(X,Y).
    • bucket bucket
  – **idiomatic meaning:**
    • word meaning
    • john john
    • kick <None>
    • bucket <None>
    • kick the bucket die(X).

humanities.byu.edu/.../kick_the_bucket.html
cf. “kick a bucket”
Today

- look in some detail at what we started last time...

**Basic DCG:**

- sentence --> np, vp.
- vp --> v, np.
- v --> [likes].
- np --> [john].
- np --> [mary].

**Query:** (we supply two arguments: sentence as a list and an empty list)

?- sentence([john,likes,mary],[[]]).
Yes (Answer)

**Phrase Structure DCG:**

- sentence(sentence(NP,VP)) --> np(NP), vp(VP).
- vp(vp(V,NP)) --> v(V), np(NP).
- v(v(likes)) --> [likes].
- np(np(john)) --> [john].
- np(np(mary)) --> [mary].

**Query:** (supply one more argument)

- ?- sentence(PS,[john,likes,mary],[[]]).
 PS = sentence(np(john),vp(v(likes),np(mary)))

How to turn a basic DCG into one that “returns” more than Yes/No
Today

• look in some detail at what we started last time...

• **Basic DCG:**
  
  sentence --> np, vp.
  
  vp --> v, np.
  
  v --> [likes].
  
  np --> [john].
  
  np --> [mary].

• **Query:** (we supply two arguments: sentence as a list and an empty list)
  
  ?- sentence([john,likes,mary],[[]]).
  
  Yes (Answer)

• **Meaning DCG:**
  
  – sentence(P) --> np(NP1), vp(P),
    {saturate1(P,NP1)}.
  
  – vp(P) --> v(P), np(NP2), {saturate2(P,NP2)}.
  
  – v(likes(X,Y)) --> [likes].
  
  – np(john) --> [john].
  
  – np(mary) --> [mary].
  
  – saturate1(P,A) :- arg(1,P,A).
  

• **Query:** (supply one more argument)
  
  ?- sentence(M,[john,likes,mary],[[]]).
  
  M = likes(john,mary)
Part 1

• Computing Phrase Structure
Representing Phrase Structure in Prolog

- We don’t directly draw trees in Prolog, but we can use an “equivalent” representation.
- **example:**

  ```prolog
  sentence(np(john), vp(v(likes), np(mary)))
  ```
Modify DCG to include Phrase Structure

- **Basic DCG:**
  
<table>
<thead>
<tr>
<th>Prolog</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>john</td>
</tr>
<tr>
<td>mary</td>
<td>mary</td>
</tr>
<tr>
<td>likes</td>
<td>likes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **DCG rules:**

  - np --&gt; [john].
  - np --&gt; [mary].

- **Procedure:**
  
  - for each DCG rule, add one argument that encodes the equivalent tree fragment

- **add one argument:**

  - np( ) --&gt; [john].
  - np( ) --&gt; [mary].

- **substitute tree fragment:**

  - np(np(john)) --&gt; [john].
  - np(np(mary)) --&gt; [mary].

Prolog Tree

<table>
<thead>
<tr>
<th>Prolog</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>v(likes)</td>
<td>vp</td>
</tr>
<tr>
<td></td>
<td>likes</td>
</tr>
<tr>
<td></td>
<td>Mary</td>
</tr>
</tbody>
</table>

sentence(np(john), vp(v(likes), np(mary)))
Modify DCG to include Phrase Structure

- Basic DCG:
  sentence --> np, vp.
  vp --> v, np.
  v --> [likes].
  np --> [john].
  np --> [mary].

- Procedure:
  - for each DCG rule, add one argument that encodes the equivalent tree fragment

- DCG rule:
  v --> [likes].

- add one argument:
  v( ) --> [likes].

- substitute tree fragment:
  v(v(likes)) --> [likes].
Modify DCG to include Phrase Structure

```
sentence(np(john), vp(v(likes), np(mary)))
```

- **DCG rule:**
  
  \[ \text{vp} \rightarrow v, \text{np}. \]

- **add one argument:**
  
  \[ \text{vp}( ) \rightarrow v, \text{np}. \]
  
  *what goes in there?*

- **well, we already have transformed v and np to take one argument:**
  
  \[
  \begin{align*}
  v(v(\text{likes})) & \rightarrow \text{[likes]}. \\
  np(np(\text{john})) & \rightarrow \text{[john]}. \\
  np(np(\text{mary})) & \rightarrow \text{[mary]}. \\
  \end{align*}
  \]

- **so we have:**
  
  \[ \text{vp}( ) \rightarrow v(X), \text{np}(Y). \]

  *can't just write* \[ \text{vp}(v(\text{likes}), np(\text{mary})) \]

  *Y could be np(\text{john}), could be np(\text{mary}).*

  *we could also (in principle) have other verbs:*

  *e.g.* \[ v(v(\text{hates})) \rightarrow \text{[hates]}. \]

- **finally:**
  
  \[ \text{vp}(\text{vp}(X, Y)) \rightarrow v(X), \text{np}(Y). \]
Modify DCG to include Phrase Structure

- DCG rule:
  \[\text{sentence} \rightarrow \text{np}, \text{vp}.\]

- add one argument:
  \[\text{sentence}(\ ) \rightarrow \text{np}, \text{vp}.\]
  *what goes in there?*

- well, we already have transformed vp and np to take one argument:
  \[
  \begin{align*}
  \text{vp}(\text{vp}(X, Y)) & \rightarrow v(X), \text{np}(Y). \\
  \text{np}(\text{np}(\text{john})) & \rightarrow [\text{john}]. \\
  \text{np}(\text{np}(\text{mary})) & \rightarrow [\text{mary}].
  \end{align*}
  \]

- so we have:
  \[\text{sentence}(\ ) \rightarrow \text{np}(X), \text{vp}(Y).\]

- finally:
  \[\text{sentence}(\text{sentence}(X, Y)) \rightarrow \text{np}(X), \text{vp}(Y).\]
Modify DCG to include Phrase Structure

• modification to include one extra argument for each DCG rule is now complete

• **Basic DCG:**
  
  sentence --> np, vp.
  vp --> v, np.
  v --> [likes].
  np --> [john].
  np --> [mary].

• **Query:** (we supply two arguments: sentence as a list and an empty list)
  
  ?- sentence([john,likes,mary],[[]]).
  Yes (Answer)

• **Phrase Structure DCG:**
  
  sentence(sentence(NP,VP)) --> np(NP), vp(VP).
  vp(vp(V,NP)) --> v(V), np(NP).
  v(v(likes)) --> [likes].
  np(np(john)) --> [john].
  np(np(mary)) --> [mary].

• **Modified Query:** (supply one more argument)
  
  ?- sentence(PS,[john,likes,mary],[[]]).
  PS = sentence(np(john),vp(v(likes),np(mary)))
Part 2

• Computing Meaning
Representing Meaning in Prolog

• We don’t need to represent trees here, but we still need to know the equivalences ...
• example:
  – John likes Mary
  – likes(john, mary)

<table>
<thead>
<tr>
<th>Equivalences:</th>
<th>Meaning</th>
<th>Word/Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>likes(john,mary)</td>
<td>john</td>
<td>John</td>
</tr>
<tr>
<td></td>
<td>mary</td>
<td>Mary</td>
</tr>
<tr>
<td></td>
<td>likes(X,Y)</td>
<td>likes</td>
</tr>
<tr>
<td></td>
<td>likes(X,mary)</td>
<td>likes Mary</td>
</tr>
<tr>
<td></td>
<td>likes(X,john)</td>
<td>likes John</td>
</tr>
<tr>
<td></td>
<td>likes(john,mary)</td>
<td>John likes Mary</td>
</tr>
</tbody>
</table>
Modify DCG to include Meaning

• Basic DCG:
  sentence --> np, vp.
  vp --> v, np.
  v --> [likes].
  np --> [john].
  np --> [mary].

• Procedure:
  – for each DCG rule, add one argument that encodes the equivalent meaning fragment

• DCG rules:
  np --> [john].
  np --> [mary].

• add one argument:
  np( ) --> [john].
  np( ) --> [mary].

• substitute meaning fragment:
  np(john) --> [john].
  np(mary) --> [mary].

Equivalences:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Word/Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>John</td>
</tr>
<tr>
<td>mary</td>
<td>Mary</td>
</tr>
<tr>
<td>likes(X,Y)</td>
<td>likes</td>
</tr>
<tr>
<td>likes(X,mary)</td>
<td>likes Mary</td>
</tr>
<tr>
<td>likes(X,john)</td>
<td>likes John</td>
</tr>
<tr>
<td>likes(john,mary)</td>
<td>John likes Mary</td>
</tr>
</tbody>
</table>
Modify DCG to include Meaning

• **Basic DCG:**
  sentence --> np, vp.
  vp --> v, np.
  v --> [likes].
  np --> [john].
  np --> [mary].

• **Procedure:**
  – for each DCG rule, add one argument that encodes the equivalent meaning fragment

• **DCG rules:**
  v --> [likes].

• **add one argument:**
  v( ) --> [likes].

• **substitute meaning fragment:**
  v(likes(X,Y)) --> [likes].

---

**Equivalences:**

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Word/Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>John</td>
</tr>
<tr>
<td>mary</td>
<td>Mary</td>
</tr>
<tr>
<td>likes(X,Y)</td>
<td>likes</td>
</tr>
<tr>
<td>likes(X,mary)</td>
<td>likes Mary</td>
</tr>
<tr>
<td>likes(X,john)</td>
<td>likes John</td>
</tr>
<tr>
<td>likes(john,mary)</td>
<td>John likes Mary</td>
</tr>
</tbody>
</table>
Modify DCG to include Meaning

• **DCG rule:**
  \[
  \text{vp} \rightarrow \text{v, np}.
  \]

• **we already have transformed v and np to take one meaning argument:**
  \[
  \text{v(likes(X,Y))} \rightarrow [\text{likes}].
  \]
  \[
  \text{np(john)} \rightarrow [\text{john}].
  \]
  \[
  \text{np(mary)} \rightarrow [\text{mary}].
  \]

• **so we have:**
  \[
  \text{vp( )} \rightarrow \text{v(Vm), np(NPm)}.
  \]
  \[
  \text{variables}
  \]
  \[
  \text{Vm} = \text{“verb meaning”}, \text{NPm} = \text{“NP meaning”}
  \]

• **we need to encode the notion of argument saturation:**
  e.g. \( \text{Vm} = \text{likes(X,Y)} \)
  \( \text{NPm} = \text{mary} \)
  we want the “VP meaning” to be
  \( \text{likes(X,mary)} \)
  i.e. argument \( Y \) gets \textit{saturated}

<table>
<thead>
<tr>
<th>Equivalences:</th>
<th>Meaning</th>
<th>Word/Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>john</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td>mary</td>
<td>Mary</td>
<td></td>
</tr>
<tr>
<td>likes(X,Y)</td>
<td>likes</td>
<td></td>
</tr>
<tr>
<td>likes(X,mary)</td>
<td>likes Mary</td>
<td></td>
</tr>
<tr>
<td>likes(X,john)</td>
<td>likes John</td>
<td></td>
</tr>
<tr>
<td>likes(john,mary)</td>
<td>John likes Mary</td>
<td></td>
</tr>
</tbody>
</table>
Argument Saturation

• we’re gonna need the Prolog built-in arg/3:
  – arg(Nth,Predicate,Argument)
  – means make Nth argument of Predicate equal to Argument

• example:
  – given predicate \( p(a, b, c) \)
  – then
    – ?- arg(1,p(a,b,c),X). X=a
    – ?- arg(2,p(a,b,c),X). X=b
    – ?- arg(3,p(a,b,c),X). X=c
    – ?- arg(4,p(a,b,c),X). No

• example:
  – given predicate \( \text{likes}(\text{john}, \text{mary}) \)
  – then
    – ?- arg(1,likes(john,mary),X). X=John
    – ?- arg(2,likes(john,mary),X). X=mary
Modify DCG to include Meaning

- we already have transformed \( v \) and \( np \) to take one meaning argument:
  \[
  v(\text{likes}(X,Y)) \rightarrow [\text{likes}].
  \]
  \[
  np(\text{john}) \rightarrow [\text{john}].
  \]
  \[
  np(\text{mary}) \rightarrow [\text{mary}].
  \]
- we have:
  \[
  \text{vp}(\text{ }) \rightarrow v(\text{Vm}), \ np(\text{NPm}).
  \]
- we need to encode the notion of argument saturation:
  e.g. \( \text{Vm} = \text{likes}(X,Y) \)
  \[
  \text{NPm} = \text{mary}
  \]
- here:
  VP meaning must be \( \text{Vm} \)
  but with \( \text{arg}(2, \text{Vm}, \text{NPm}) \) being true

\( i.e. \) 2nd argument of \( \text{Vm} \) (namely \( Y \)) must be the NP meaning
Modify DCG to include Meaning

- we need to encode the notion of *argument saturation*:
  e.g. \( V_m = \text{likes}(X,Y) \)
  \( N_{Pm} = \text{mary} \)
  VP meaning must be \( V_m \)
  but with \( \text{arg}(2, V_m, N_{Pm}) \) being true

- we then have:
  \( \text{vp}(V_m) \rightarrow v(V_m), n_p(N_{Pm}), \{\text{saturate2}(V_m, N_{Pm})\} \).

- New notation: “*curly braces*”
  - \( \{<\text{Goal}>\} \) means call Prolog \( <\text{Goal}> \)
  - \( \{\text{arg}(2, V_{Bm}, N_{Pm})\} \) means call \( \text{arg}(2, V_{Bm}, N_{Pm}) \)

- perhaps more clearly, we can re-write our DCG rule as:
  \( \text{vp}(V_m) \rightarrow v(V_m), n_p(N_{Pm}), \{\text{saturate2}(V_m, N_{Pm})\} \).

- and define the rule (in the Prolog database):
  \( \text{saturate2}(P,A) :- \text{arg}(2,P,A) \).
Modify DCG to include Meaning

- **finally:**
  
  sentence --> np, vp.

- **we already have transformed vp and np to take one meaning argument:**

  \[
  \begin{align*}
  \text{vp}(V_m) & \rightarrow v(V_m), \text{np}(N_P,m), \{\text{saturate2}(V_m,N_Pm)\}. \\
  \text{np}(\text{john}) & \rightarrow [\text{john}]. \\
  \text{np}(\text{mary}) & \rightarrow [\text{mary}]. 
  \end{align*}
  \]

- **we need to encode the notion of argument saturation:**

  e.g. \(V_m = \text{likes}(X,\text{mary})\)
  \[\text{NPm} = \text{john}\]

  we want the “sentence meaning” to be
  \(\text{likes}(\text{john},\text{mary})\)

  i.e. 1st argument \(X\) gets saturated

- **we then have:**

  sentence(V_P,m) --> np(N_P,m), vp(V_P,m),
  \{arg(1,V_P,m,N_Pm)\}.
Modify DCG to include Meaning

- we are done...

**Basic DCG:**

```prolog
sentence --> np, vp.
vp --> v, np.
v --> [likes].
np --> [john].
np --> [mary].
```

**Query:** (we supply two arguments: sentence as a list and an empty list)

```prolog
?- sentence([john,likes,mary],[]).
Yes (Answer)
```

**Meaning DCG:**

```prolog
sentence(P) --> np(NP1), vp(P), {saturate1(P,NP1)}.
vp(P) --> v(P), np(NP2), {saturate2(P,NP2)}.
v(likes(X,Y)) --> [likes].
np(john) --> [john].
np(mary) --> [mary].
saturate1(P,A) :- arg(1,P,A).
saturate2(P,A) :- arg(2,P,A).
```

**Query:** (supply one more argument)

```prolog
?- sentence(M,[john,likes,mary],[]).
M = likes(john,mary)
```

You now know how to turn a basic DCG into one that "returns" the meaning of a sentence.
Exercise

• **Basic DCG for practice (use menu File -> New to create a file):**

```
sentence --> np, vp.
vp --> v, np.
v --> [likes].
v --> [hates].
np --> det, n.
np --> [john].
np --> [mary].
det --> [the].
det --> [a].
n --> [book].
```

• **Sentences:**
  - John hates the book
  - John likes mary

• **Phrase Structures:**
  - sentence(np(john), vp(v(hates), np(det(the), n(book))))
  - sentence(np(john), vp(v(likes), np(mary)))

• **Meanings:**
  - hates(john, book).
  - likes(john, mary).