LING 364: Introduction to Formal Semantics

Lecture 12

February 21st

Administrivia

- Reminder
  - Computer Lab Class on Thursday
  - meet in Social Sciences 224 (not here)
  - Homework 3 will be given out

Administrivia

- Reading for Thursday
  - Chapter 4: Modifiers
  - *already given out earlier with Chapter 3*
  - no quiz: but will be part of your homework

  - *we will start looking at Chapter 4 today with adjectives*

Homework 2 Review

Homework 2 Review

- Exercises 1 through 3
  - Give a **basic** DCG grammar for the following examples:
  - [$_{Sbar}$[$_{S}$ [$_{NP}$ John] [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]]
  - [$_{Sbar}$[$_{S}$ [$_{NP}$ Pete] [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]]
  - [$_{Sbar}$[$_{S}$ [$_{NP}$ Mary] [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]
  - [$_{Sbar}$[$_{S}$ [$_{NP}$ Pete] [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]
  - [$_{Sbar}$[$_{S}$ [$_{NP}$ John] [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{NEG}$ not] [$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is][$_{NP}$ [$_{NEG}$ not] [$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is] [$_{NP}$[$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]][$_{CONJ}$ and][$_{NP}$ [$_{DET}$ a][$_{N}$ baseball fan]]]]]]
  - [$_{Sbar}$ [$_{NP}$ Who] [$_{S}$ [$_{VP}$ [$_{V}$ is] [$_{NP}$[$_{NP}$ [$_{DET}$ a][$_{N}$ student]]]]][$_{CONJ}$ and][$_{NP}$ [$_{NEG}$ not][$_{NP}$[$_{DET}$ a][$_{N}$ baseball fan]]]]]]]

Homework 2 Review

- Basic DCG
  - i.e. no phrase structure or meaning computed, just Yes/No answers from query
  - `?- sbar(Sentence,[]).`
  - `Yes/No`
- Grammar rules
- `sbar --> np, s.`
- `sbar --> s.`
- `s --> vp.`
- `s --> np, vp.`
- `np --> [john].`
- `np --> [pete].`
- `np --> [mary].`
- `np --> det, n.`
- `np --> [who].`
- `np --> neg, np.`
- `np --> np, conj, np.`
- `n --> [student].`
- `n --> [baseball,fan].`
- `neg --> [not].`
- `conj --> [and].`
- `vp --> v, np.`
- `v --> [is].`
- `det --> [a].`

Homework 2 Review

- Exercise 4
  - Modify the grammar to include phrase structure

```
| ?- sbar(PS,[who,is,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(neg(not),np(det(a),n(baseball_fan)))))) ?
| ?- sbar(PS,[john,is,a,baseball,fan],[]).
PS = sbar(s(np(john),vp(v(is),np(det(a),n(baseball_fan))))) ?
| ?- sbar(PS,[who,is,a,student,and,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(det(a),
n(baseball_fan)))))) ?
| ?- sbar(PS,[who,is,a,student,and,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(neg(not),
np(det(a),n(baseball_fan))))))) ?
```

Homework 2 Review

- Modify basic DCG into one that includes phrase structure

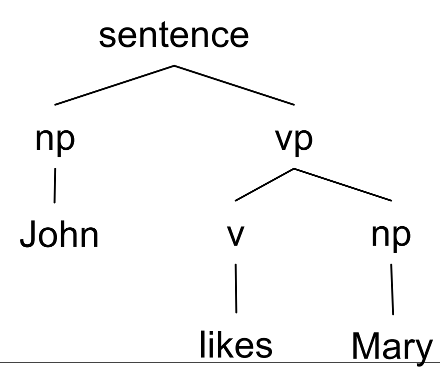
- **Basic DCG:**

```
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)

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



- **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)))
```

Homework 2 Review

- **Step 1**:
  - *add phrase structure for each rule*
- sbar(**sbar(NP,S)**) --> np(**NP**), s(**S**).
- sbar(**sbar(S)**) --> s(**S**).
- s(**s(VP)**) --> vp(**VP**).
- s(**s(NP,VP)**) --> np(**NP**), vp(**VP**).
- np(**np(who)**) --> [who].
- np(**np(john)**) --> [john].
- np(**np(pete)**) --> [pete].
- np(**np(mary)**) --> [mary].
- np(**np(Det,N)**) --> det(**Det**), n(**N**).
- np(**np(Neg,NP)**) --> neg(**Neg**), np(**NP**).
- np(**np(NP1,Conj,NP2)**) --> np(**NP1**), conj(**Conj**), np(**NP2**).
- neg(**neg(not)**) --> [not].
- conj(**conj(and)**) --> [and].
- vp(**vp(V,NP)**) --> v(**V**), np(**NP**).
- v(**v(is)**) --> [is].
- det(**det(a)**) --> [a].
- n(**n(student)**) --> [student].
- n(**n(baseball_fan)**) --> [baseball,fan].

Homework 2 Review

- **Step 1**:
  - *add phrase structure for each rule*
- sbar(**sbar(NP,S)**) --> np(**NP**), s(**S**).
- sbar(**sbar(S)**) --> s(**S**).

- Problem:
- ?- sbar(X,[who,is,a,student],[]).
- X = sbar(np(who),s(vp(v(is),np(det(a),n(student)))))

- | ?- sbar(X,[john,is,a,student],[]).
- X = sbar(np(john),s(vp(v(is),np(det(a),n(student))))) ?

Homework 2 Review

- **Step 1**:
  - *add phrase structure for each rule*
- Flipping the rule order doesn't help
- sbar(**sbar(S)**) --> s(**S**). sbar(**sbar(NP,S)**) --> np(**NP**), s(**S**).

- Problem:
- ?- sbar(X,[john,is,a,student],[]).
- X = sbar(s(np(john),vp(v(is),np(det(a),n(student)))))

- | ?- sbar(X,[who,is,a,student],[]).
- X = sbar(s(np(who),vp(v(is),np(det(a),n(student)))))

Homework 2 Review

- **Step 2**:
  - *need to separate* who *from other noun phrases*
  - **Solution**: realize you can rename a non-terminal and still return the same phrase
- sbar(**sbar(S)**) --> s(**S**).
  sbar(**sbar(NP,S)**) --> **wh_np**(**NP**), s(**S**).
- **wh_np**(np(who)) --> [who].

- Correct output:
- ?- sbar(X,[who,is,a,student],[]).
- X = sbar(np(who),s(vp(v(is),np(det(a),n(student))))) ?
- | ?- sbar(X,[john,is,a,student],[]).
- X = sbar(s(np(john),vp(v(is),np(det(a),n(student)))))

Homework 2 Review

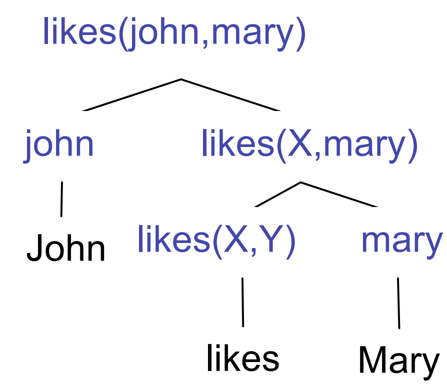
- Exercise 5
  - Modify the grammar to generate meaning

```
| ?- sbar(M,[who,is,not,a,baseball,fan],[]).
M = \+baseball_fan(_A) ?
| ?- sbar(M,[john,is,a,baseball,fan],[]).
M = baseball_fan(john) ?
| ?- sbar(M,[who,is,a,student,and,a,baseball,fan],[]).
M = student(_A),baseball_fan(_A) ?
| ?- sbar(M,[who,is,a,student,and,not,a,baseball,fan],[]).
M = student(_A),\+baseball_fan(_A) ?
```

Note:
_A is an internally-generated Prolog variable

Homework 2 Review

- modify basic DCG into one that includes meaning



- **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)

argument saturation

`arg(Nth,Predicate,Argument)` means make `Nth` argument of `Predicate` equal to `Argument`

`{ <Goal> }` means call Prolog `<Goal>`
`{arg(2,VBm,NPm)}` means call `arg(2,VBm,NPm)`

- **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).`
  - `saturate2(P,A) :- arg(2,P,A).`
- **Query**: (supply one more argument)
- `?- sentence(M,[john,likes,mary],[]).`

  `M = likes(john,mary)`

Homework 2 Review

- **Step 1**:
  - *add meaning for each rule*
- ***note****: we don't have to do the* wh_np *renaming here*
- sbar(**P**) --> np(**x**), s(**P**), {saturate1(**P**,**x**)}.
- sbar(**P**) --> s(**P**).
- s(**P**) --> vp(**P**).
- s(**P**) --> np(**X**), vp(**P**), {saturate1(**P**,**X**)}.
- np(**john**) --> [john].
- np(**pete**) --> [pete].
- np(**mary**) --> [mary].
- np(**P**) --> det(**a**), n(**P**).
- np(**(\+ P)**) --> neg, np(**P**).
- np(**(P1,P2)**) --> np(**P1**), conj(**and**), np(**P2**).
- np(**x**) --> [who].
- neg --> [not].
- conj(**and**) --> [and].
- vp(**P**) --> v(**copula**), np(**P**).
- v(**copula**) --> [is].
- det(**a**) --> [a].
- n(**student(X)**) --> [student].
- n(**baseball_fan(X)**) --> [baseball,fan].

Homework 2 Review

- **Step 2**:
  - *generalize saturate1/2 to work with logical connectives like \+ and ,*
- sbar(**P**) --> np(**x**), s(**P**), {saturate1(**P**,**x**)}.
- sbar(**P**) --> s(**P**).
- s(**P**) --> vp(**P**).
- s(**P**) --> np(**X**), vp(**P**), {saturate1(**P**,**X**)}.
- np(**john**) --> [john].
- np(**pete**) --> [pete].

- Redefine:
  - `saturate1((P1,P2),X) :- saturate1(P1,X), saturate1(P2,X).`
  - `saturate1((\+ P),X) :- saturate1(P,X).`
  - `saturate1(P,X) :- arg(1,P,X).`

Lambda Calculus

- *Two lectures ago...*
- Basic mechanisms
- **lambda expression**
  - **variable substitution**
- **variable substitution**
  - *aka* **Beta (β)-reduction**
  - **“cut-and-paste”**
- **variable renaming**
  - *aka* **Alpha (α)-reduction**
  - **to avoid variable name clashes**
  - e.g. **“rename x’s to y’s”**

likes `likes(X,Y).`

likes [λy.[λx.x likes y]]

[λy.[λx.x likes y]](Mary)

⇩

[λx.x likes **y**]   [λy.](**Mary**)

⇩

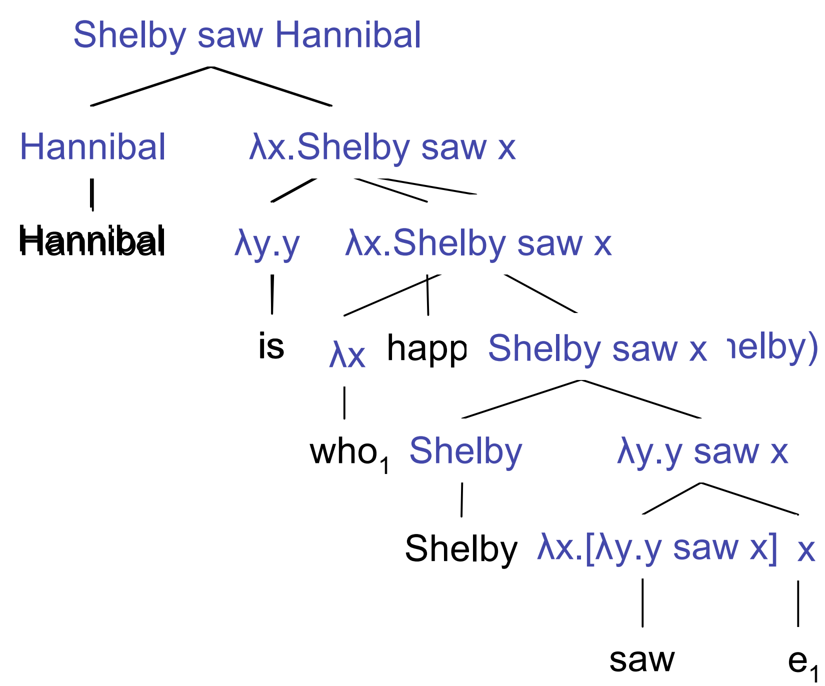
[λx.x likes **Mary**]

λx.x likes Mary

λy.y likes Mary

Lambda Calculus

- Relative Clauses (also Topicalization)
  - (7) Hannibal is [who Shelby saw]
  - [who Shelby saw] has meaning λx.Shelby saw x



Chapter 4: Modifiers

Chapter 4: Modifiers

- Examples:
  - (1) Ossie is a bird `bird(ossie).`
    - *bird*: predicative nominal
  - (2) Ossie is tall `tall(ossie).`
    - *tall*: predicative adjective
  - (3) Ossie is a tall bird
    - tall: attributive adjective (modifies noun *bird*)

    - what is the semantics of (3)?

Chapter 4: Modifiers

- Example:
  - (3) Ossie is a tall bird
- One view (intersective):
  - *tall* `tall(X).`
  - *bird* `bird(X).`
  - *tall bird* `tall(X), bird(X).`
  - *Ossie is a tall bird* `tall(ossie), bird(ossie).`

- How do we encode this in the lambda calculus?

Chapter 4: Modifiers

- Example:
  - (3) Ossie is a tall bird
- Problems with the intersective viewpoint:
  - `tall(X)`: set of things that are tall, say, T
  - `bird(X)`: set of birds, say, B
  - `tall(X), bird(X)`: intersection, so T ∩ B.

set intersection

But isn't tall a relative concept?
e.g. *tall bird = tall for a bird*
(*cf. dead as in dead bird*)

Not all adjectives are intersective
e.g. *former* as in *former teacher*

Chapter 4: Modifiers

- Example:
  - (3) Ossie is a tall bird
- Another viewpoint (roughly):
  - (diagram 23 in Chapter 4)
  - *tall* λp.[λx.[p x & x is taller_than p average]]
  - *bird* bird
  - *tall bird* [λp.[λx.[p x & x is taller_than p average]]](bird)
  - λx. bird x & x is taller_than bird average
  - *Ossie is a tall bird*
  - [λx. bird x & x is taller_than bird average](Ossie)
  - bird Ossie & Ossie is taller_than bird average