LING 364: Introduction to Formal Semantics

Lecture 7

February 2nd

Administrivia

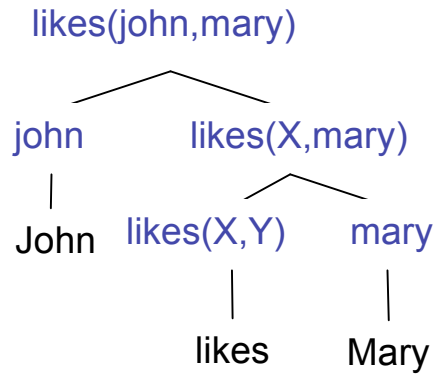
- **today**
  - (3:30pm – 4:40pm)
    - lecture here in Comm 214
  - (4:45pm – 5:45pm) (**EXTRA)**
    - lab practice in Social Sciences Lab 224

- **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

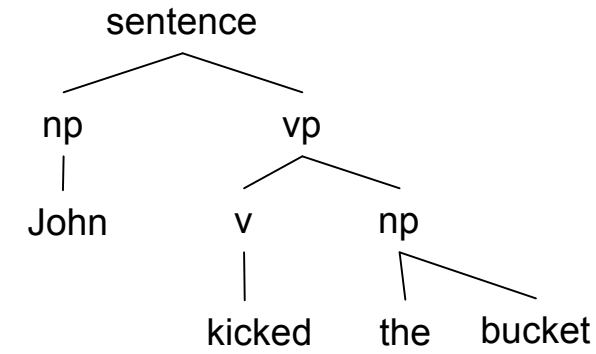
word or phrase	**meaning**
John	john
likes Mary	likes(X,mary).
likes	likes(X,Y).
Mary	mary

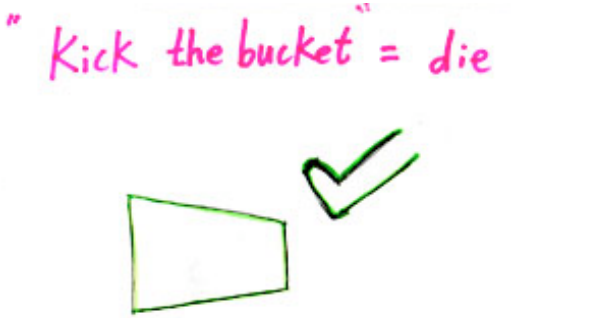


  - each word here has a contribution to make to the meaning of the complete sentence
  - cf. it is raining (pleonastic “it”/ambient “it”)

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)
```

How to turn a basic DCG into one that "returns" the meaning of a sentence

- **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)
```

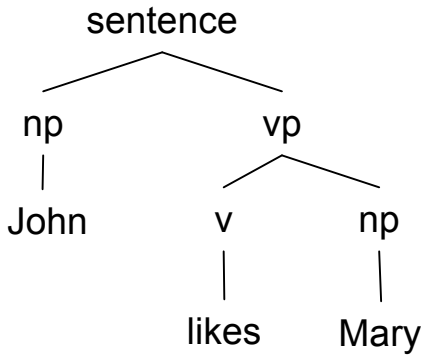
Part 1

- Computing Phrase Structure

Representing Phrase Structure in Prolog

- We don't directly draw trees in Prolog, but we can use an "equivalent" representation
- **example**:

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



Notation:

Prolog	Tree
`john`	`john`
`mary`	`mary`
`likes`	`likes`
`np(john)`	np → John
`np(mary)`	np → Mary

Prolog	Tree
`v(likes)`	v → likes
`vp(v(likes),np(mary))`	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 rules:**

```
np --> [john].
np --> [mary].
```

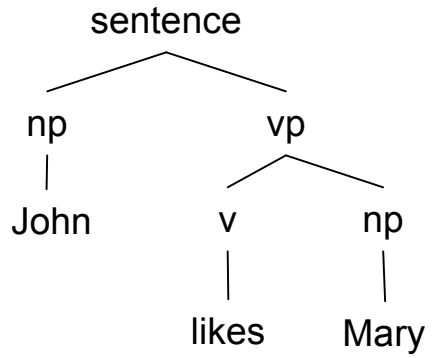
- **add one argument:**

```
np( ) --> [john].
np( ) --> [mary].
```

- **substitute tree fragment:**

```
np(np(john)) --> [john].
np(np(mary)) --> [mary].
```

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



Prolog	Tree
john	john
mary	mary
likes	likes
np(john)	np / John
np(mary)	np / Mary

Prolog	Tree
v(likes)	v / likes
vp(v(likes),np(mary))	vp / v, np / likes, 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].
```

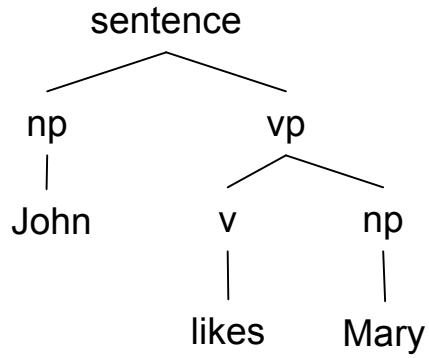
- **add one argument:**

```
v( ) --> [likes].
```

- **substitute tree fragment:**

```
v(v(likes)) --> [likes].
```

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

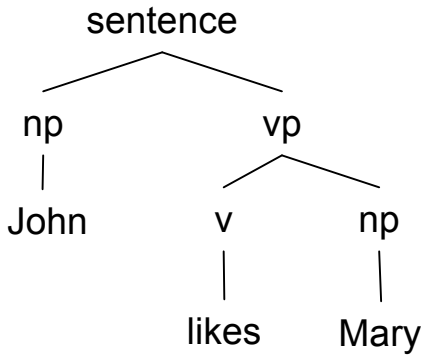


Prolog	Tree
john	john
mary	mary
likes	likes
np(john)	np / John
np(mary)	np / Mary

Prolog	Tree
v(likes)	v / likes
vp(v(likes),np(mary))	vp / v, np / likes, Mary

Modify DCG to include Phrase Structure

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



- **DCG rule:**

  ```
  vp --> v, np.
  ```

- **add one argument:**

  ```
  vp( ) --> v, np.
  ```

  what goes in there?

- **well, we already have transformed v and np to take one argument:**

  ```
  v(v(likes)) --> [likes].
  np(np(john)) --> [john].
  np(np(mary)) --> [mary].
  ```

- **so we have:**

  ```
  vp(●) --> v(X), np(Y).
  ```

  can't just write `vp(v(likes),np(mary))`

  `Y` *could be* `np(john)`, *could be* `np(mary)`

  we could also (in principle) have other verbs:

  e.g. `v(v(hates)) --> [hates].`

- **finally:**

  ```
  vp(vp(X,Y)) --> v(X), np(Y).
  ```

Prolog	Tree
`john`	`john`
`mary`	`mary`
`likes`	`likes`
`np(john)`	np → John
`np(mary)`	np → Mary

Prolog	Tree
`v(likes)`	v → likes
`vp(v(likes),np(mary))`	vp → (v → likes, np → Mary)

Modify DCG to include Phrase Structure

- **DCG rule:**

```
sentence --> np, vp.
```

- **add one argument:**

```
sentence( ) --> np, vp.
```

what goes in there?

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

```
vp(vp(X,Y)) --> v(X), np(Y).
np(np(john)) --> [john].
np(np(mary)) --> [mary].
```

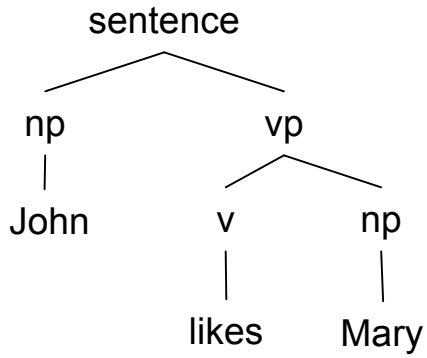
- **so we have:**

```
sentence( ) --> np(X),vp(Y).
```

- **finally:**

```
sentence(sentence(X,Y)) -->
   np(X), vp(Y).
```

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



Prolog	Tree
john	john
mary	mary
likes	likes
np(john)	np
	John
np(mary)	np
	Mary

Prolog	Tree
v(likes)	v
	likes
vp(v(likes),np(mary))	vp
	v
	likes
	np
	Mary

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)

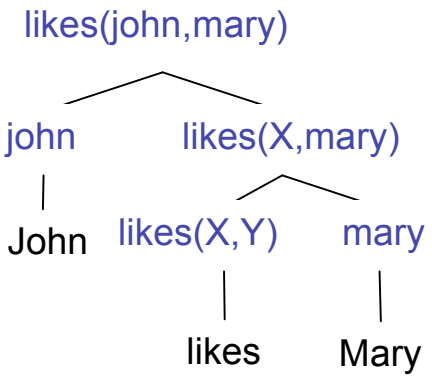
```
?- sentence(PS,[john,likes,mary],[]).
PS = sentence(np(john),vp(v(likes),np(mary)))
```

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)`



Equivalences:

Meaning	**Word/Phrase**
`john`	`John`
`mary`	`Mary`
`likes(X,Y)`	`likes`
`likes(X,mary)`	`likes Mary`
`likes(X,john)`	`likes John`
`likes(john,mary)`	`John likes Mary`

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:**

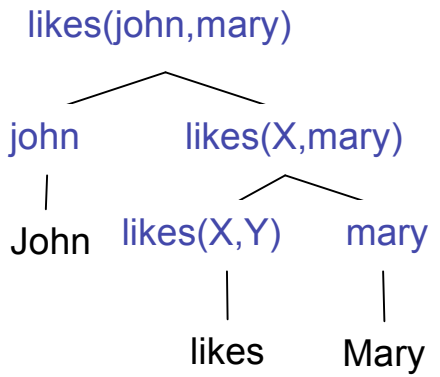
```
np --> [john].
np --> [mary].
```

- **add one argument:**

```
np( ) --> [john].
np( ) --> [mary].
```

- **substitute meaning fragment:**

```
np(john) --> [john].
np(mary) --> [mary].
```



Equivalences:

Meaning	Word/Phrase
john	John
mary	Mary
likes(X,Y)	likes
likes(X,mary)	likes Mary
likes(X,john)	likes John
likes(john,mary)	John likes Mary

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:**

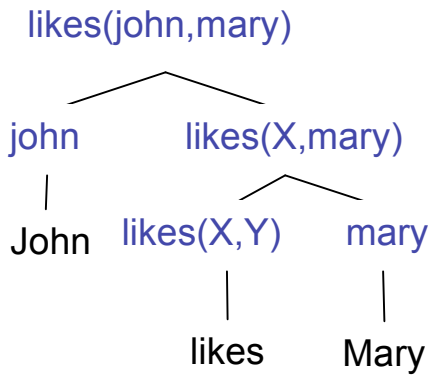
```
v --> [likes].
```

- **add one argument:**

```
v( ) --> [likes].
```

- **substitute meaning fragment:**

```
v(likes(X,Y)) --> [likes].
```



Equivalences:

Meaning	Word/Phrase
john	John
mary	Mary
likes(X,Y)	likes
likes(X,mary)	likes Mary
likes(X,john)	likes John
likes(john,mary)	John likes Mary

Modify DCG to include Meaning

- **DCG rule:**

```
vp --> v, np.
```

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

```
v(likes(X,Y)) --> [likes].
np(john) --> [john].
np(mary) --> [mary].
```

- **so we have:**

```
vp( ) --> v(Vm), np(NPm).
```

variables

`Vm` = "verb meaning", `NPm` = "NP meaning"

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

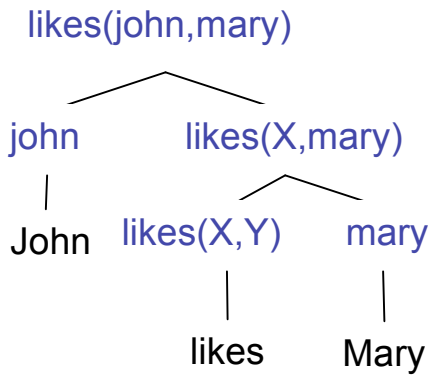
e.g. `Vm = likes(X,Y)`

`NPm = mary`

we want the "VP meaning" to be

`likes(X,mary)`

i.e. argument `Y` gets *saturated*



Equivalences:

Meaning	Word/Phrase
`john`	`John`
`mary`	`Mary`
`likes(X,Y)`	`likes`
`likes(X,mary)`	`likes Mary`
`likes(X,john)`	`likes John`
`likes(john,mary)`	`John likes Mary`

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 `likes(john,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(likes(X,Y)) --> [likes].
np(john) --> [john].
np(mary) --> [mary].
```

- **we have:**

```
vp( ) --> v(Vm), np(NPm).
```

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

  e.g. `Vm = likes(X,Y)`

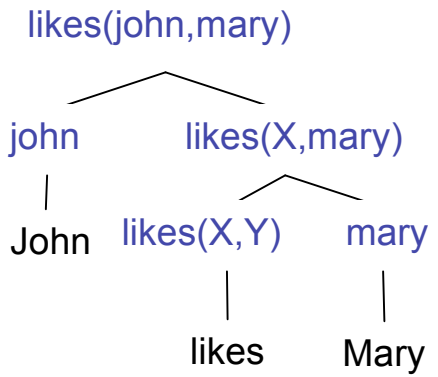
  `NPm = mary`

- **here:**

  VP meaning must be `Vm`

  but with `arg(2,Vm,NPm)` being true

  i.e. 2nd argument of `Vm` (namely `Y`) must be the NP meaning



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

Modify DCG to include Meaning

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

  e.g. `Vm = likes(X,Y)`

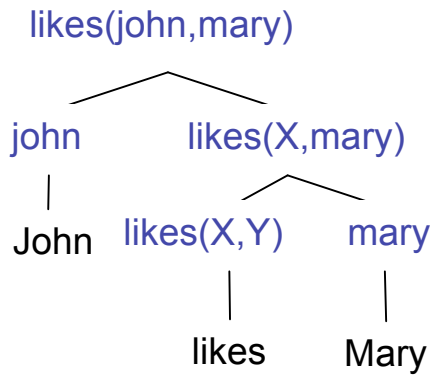
  `NPm = mary`

  VP meaning must be `Vm`

  but with `arg(2,Vm,NPm)` being true

- **we then have:**

```
vp(Vm) --> v(Vm), np(NPm), {arg(2,VBm,NPm)}.
```



- **New notation: "*curly braces*"**
- `{ <Goal> }` means call Prolog `<Goal>`
- `{arg(2,VBm,NPm)}` means call `arg(2,VBm,NPm)`

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

- **perhaps more clearly, we can re-write our DCG rule as:**

```
vp(Vm) --> v(Vm), np(NPm), {saturate2(Vm,NPm)}.
```

- **and define the rule (in the Prolog database):**

```
saturate2(P,A) :- 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:**

```
vp(Vm) --> v(Vm), np(NPm), {saturate2(Vm,NPm)}.
np(john) --> [john].
np(mary) --> [mary].
```

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

e.g. `Vm = likes(X,mary)`

`NPm = john`

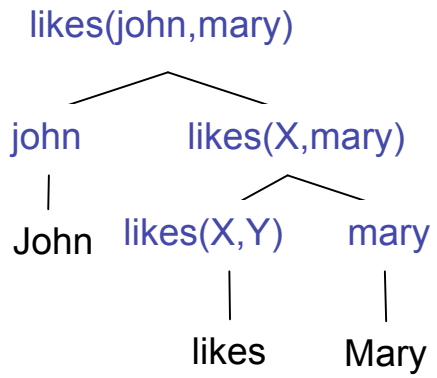
we want the "sentence meaning" to be

`likes(john,mary)`

i.e. 1st argument `X` gets *saturated*

- **we then have:**

```
sentence(VPm) --> np(NPm), vp(VPm),
   {arg(1,VPm,NPm)}.
```



`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)`

Modify DCG to include Meaning

- we are done...

- **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).`
  - `saturate2(P,A) :- arg(2,P,A).`
- **Query**: (supply one more argument)
- `?- 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).`