# LING 364: Introduction to Formal Semantics

Lecture 21 April 4th

# Administrivia

### Homework 3

- graded and returned
- homework 4 should be coming back this week as well

# Administrivia

## this Thursday

- computer lab class
- fun with quantifiers... homework 5
- meet in SS 224

# Today's Topic

## Continue with

- Reading Chapter 6: Quantifiers
- Quiz 5 (end of class: postponed)

## **Last Time**

#### Quantified NPs:

- "something to do with indicating the quantity of something"
- every child, nobody
- two dogs, several animals
- most people
- 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

(6)	every baby	exactly one baby	most babies
cried	1		<b>✓</b>
jumped		1	
swam			1

### Generalized quantifiers:

sets of sets property = set

## **Last Time**

- Defining every\_baby(P)?
- (Montague-style)
- every\_baby(P) is shorthand for
  - λP.[∀X.[baby(X) -> P(X)]]
  - ∀: for all (universal quantifier: logic symbol)
- Example:
  - every baby walks
  - [<sub>NP</sub> every baby] [<sub>VP</sub> walks]
    - λP.[∀X.[baby(X) -> P(X)]] (walks)
    - $\forall X.[baby(X) -> walks(X)]$
- Prolog-style:
- ?- \+ (baby(X), \+ walks(X)). "it's not true that there is a baby (X) who doesn't walk"

# Conversion to Prolog form

- Show
  - $\forall X.$ [baby(X) -> walks(X)]

translate this

need to

- is equivalent to (can be translated into):
  - ?- \+ (baby(X), \+ walks(X)).

We're going to use the idea that  $\forall X P(X)$  is the same as  $\neg \exists X \neg P(X)$  let's call this the "**no exception**" idea

∃= "there exists" (quantifier)

(implicitly: all Prolog variables are existentially quantified variables)

a Prolog variable like X in this query has the meaning:

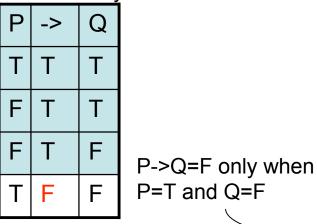
"give me some value of X such that baby(X) is true"

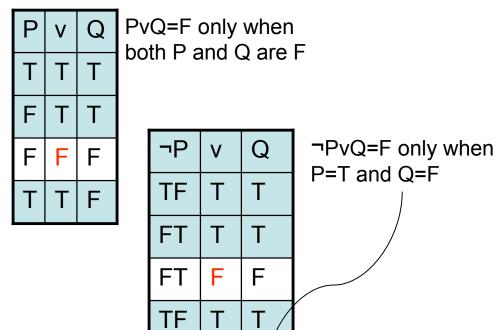
i.e. "give me some X"

i.e.  $\exists X \text{ baby}(X)$ 

# Aside: Truth Tables

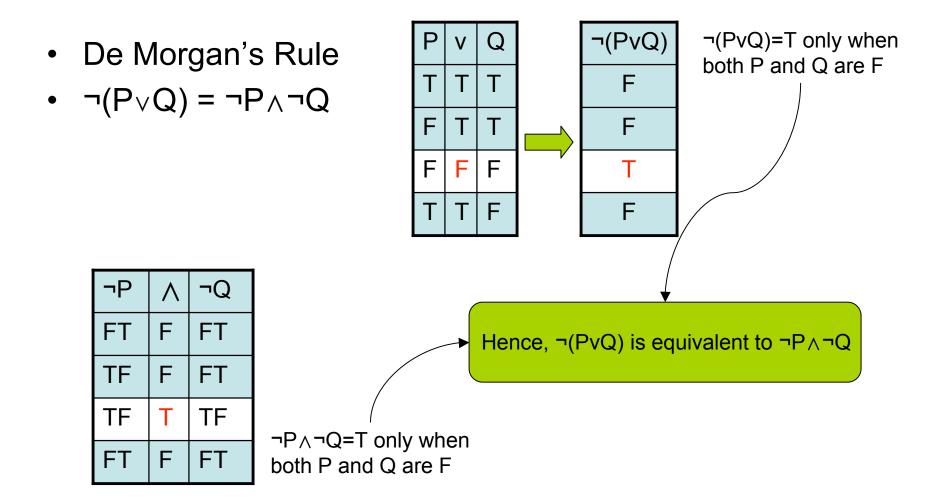
- logic of implication
- P -> Q = (*truth value*)
- T T T
- F T T
- F F T
- T F F
- i.e. if P is true, Q must be true in order for P->Q to be true
- if P is false, doesn't matter what Q is, P->Q is true
- conventionally written as:





Hence, P->Q is equivalent to ¬PvQ

# Aside: Truth Tables



# Conversion into Prolog

Note:  $\+ (baby(X), \+walks(X))$  is Prolog for  $\forall X (baby(X) -> walks(X))$ 

#### Steps:

- ¬ X (baby(X) -> walks(X))
- → X (¬baby(X) v walks(X))
  - (since P->Q = ¬PvQ, see truth tables from two slides ago)
- $\neg \exists X \neg (\neg baby(X) \lor walks(X))$ 
  - (since  $\forall X P(X) = \neg \exists X \neg P(X)$ , no exception idea from 3 slides ago)
- ¬∃X (baby(X) ∧¬walks(X))
  - (by De Morgan's rule, see truth table from last slide)
- ¬(baby(X) ∧¬walks(X))
  - (can drop ∃ X since all Prolog variables are basically existentially quantified variables)
- \+ (baby(X) ∧ \+walks(X))
  - (\+ = Prolog negation symbol)
- \+ (baby(X), \+walks(X))
  - (, = Prolog conjunction symbol)

## Last Time

- Defining every\_baby(P)?
- (Montague-style) λP.[∀X.baby(X) -> 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: baby(X)} set of all X such that baby(X) is true
- {X: walks(X)} set of all X such that walks(X) is true
- Subset relation (⊆)
- {X: baby(X)} ⊆{X: walks(X)} the "baby" set must be a subset of the "walks" set

- Imagine a possible world:
  - baby(a).
  - baby(b).
  - baby(c).
  - walks(a).
  - walks(b).
  - walks(c).
  - walks(d).
  - $\{a,b,c\} \subseteq \{a,b,c,d\}$
  - baby ⊆ walks

# Subset and Prolog

- How to express this as a Prolog query?
- Findall/3 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

```
Also need a Prolog definition of the subset relation. For example: subset([],\_). \qquad \text{``empty set is a subset of anything''} \\ subset([X|L1],L2):-member(X,L2), subset(L1,L2). \\ member(X,[X|\_]). \\ member(X,[L]):-member(X,L).
```

#### **Prolog Head-Tail List Notation:**

```
[a,b,c]
a is the head of the list (the first element)
[b,c] is the tail of the list (all but the first element)
we can write a list as follows:
```

```
[head | tail]
[a | [b,c] ]
```

programmatically: [ X | L1] will match [a,b,c] when X = a, L1 = [b,c]

# Generalized Quantifiers

- Example: every baby walks
- $\{X: baby(X)\} \subseteq \{X: walks(X)\}$  the "baby" set must be a subset of the "walks" set
- Assume the following definitions are part of the database:

```
subset([],\_).\\ subset([X|L1],L2) :- member(X,L2), subset(L1,L2).\\ member(X,[X|\_]).\\ member(X,[-],L]) :- member(X,L).
```

- 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).

L1 = [a,b]L2 = [a,b,c]

?- subset(L1,L2) is true

- False for world:
  - baby(a).baby(b).baby(d).
  - walks(a).walks(b).walks(c).

L1 = [a,b,d] L2 = [a,b,c] ?- subset(L1,L2) is false

# Generalized Quantifiers

- Example: every baby walks
- (Barwise & Cooper-style)  $\{X: baby(X)\} \subseteq \{X: walks(X)\}$
- how do we define every\_baby(P)?
- (Montague-style)  $\lambda P.[\forall X (baby(X) -> P(X))]$
- (Barwise & Cooper-style)  $\{X: baby(X)\} \subseteq \{X: P(X)\}$
- how do we define every?
- (Montague-style)  $\lambda P_1 \cdot [\lambda P_2 \cdot [\forall X (P_1(X) \rightarrow P_2(X))]]$
- (Barwise & Cooper-style)  $\{X: P_1(X)\} \subseteq \{X: P_2(X)\}$

- how do we define the expression every?
- (Montague-style)  $\lambda P_1.[\lambda P_2.[\forall X (P_1(X) \rightarrow P_2(X))]]$
- Let's look at computation in the lambda calculus...
- Example: every man likes John

```
Word ExpressionΔΡ [λΡ [∀Υ]
```

- every  $\lambda P_1.[\lambda P_2.[\forall X (P_1(X) \rightarrow P_2(X))]]$ 

*– man* man

- likes  $\lambda Y.[\lambda X.[X likes Y]]$ 

*– John* John

• **Syntax**: [<sub>S</sub> [<sub>NP</sub> [<sub>Q</sub> every][<sub>N</sub> man]][<sub>VP</sub> [<sub>V</sub> likes][<sub>NP</sub> John]]]

• Example: [S [NP [Q every][N man]][VP [V likes][NP John]]]

 $\begin{array}{lll} - & \textit{Word} & \textit{Expression} \\ - & \textit{every} & \lambda P_1.[\lambda P_2.[\,\forall\, X\, (P_1(X) \rightarrow P_2(X))]] \\ - & \textit{man} & \text{man} \\ - & \textit{likes} & \lambda Y.[\lambda X.[\,\, X\, \text{likes}\,\, Y]] \\ - & \textit{John} & \text{John} \end{array}$ 

• Steps:

 $\forall X (man(X) \rightarrow \lambda X.[X likes John](X))$ 

**Example:**  $[_{S}[_{NP}[_{Q} \text{ every}]]_{N} \text{ man}]][_{VP}[_{V} \text{ likes}]]_{NP} \text{ John}]]]$ Word **Expression** extra parentheses \+ (P1, \+ P2). every needed here man(X). man I've cheated a bit here... likes likes(X,Y). this X is the same X as the X in man(X)... John john in a program I would have to also saturate both to the same variable Steps (Prolog-style): ?- Q = (\+ (P1,\+P/2)), N= man(X), arg(1,E,C), saturate1(C,N). [o every][n man] NP = + (man(X)/+ P2). $[_{NP} [_{O} \text{ every}][_{N} \text{ man}]]]$ (pass up saturated Q as the value for the NP) ?- V = Iikes(X,Y), NP = john, saturate2(V,NP). [v likes][NP John] VP = likes(X, john). $[_{VP} [_{V} likes][_{NP} John]]$ (pass up saturated V as the value for the VP) [NP] [O] = [O] = [NP] [O] = [O] =?- NP = (\+ (man(X), + P2)), VP = likes(X,john), arg(1,NP,C), arg(2,C,Neg),arg(1,Neg,VP).  $[S_{NP}][N_{P}$ S = + (man(X), + likes(X, john))

(pass up **saturated NP** as the value for S)

**Example:**  $[_S [_{NP} [_Q \text{ every}]]_N \text{ man}]][_{VP} [_V \text{ likes}]]_{NP} \text{ John}]]]$ Set theory version Word **Expression** findall(U,P1,L1),findall(V,P2,L2),subset(L1,L2). every man(M). man likes likes(A,B). John john Steps: [ $_{\Omega}$  every][ $_{N}$  man]] Q= (findall(U,P1,L1),findall(V,P2,L2),subset(L1,L2)), N = man(M), arg(1,Q,FA1),arg(2,FA1,N), saturate1(FA1,X), saturate1(N,X).  $[_{NP}[_{O} \text{ every}][_{N} \text{ man}]]]$ NP = findall(X,man(X),L1),findall(V,P2,L2),subset(L1,L2)(pass up saturated Q as the value for the NP) [V likes][ND John] V=likes(A,B), NP=john, saturate2(V,NP). [VP [V likes][NP John]] VP = likes(A,john) (pass up saturated V as the value for the VP) [NP] [O] = [NP] [NP] [O] = [NP] [O] = [O] [NP] [O] = [O] [O] = [O] [O] = [O]NP = (findall(X,man(X),L1),findall(V,P2,L2),subset(L1,L2)), VP = likes(A,john),arg(2,NP,C2), arg(1,C2,FA2), arg(2,FA2,VP), saturate1(FA2,Y), saturate1(VP,Z).

**Example:** [S [NP [Q every][N man]][VP [V likes][NP John]]] Word **Expression** findall(U,P1,L1),findall(V,P2,L2),subset(L1,L2). every man(M). man likes likes(A,B). John john Steps:

```
[NP [O \text{ every}]][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}]][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}][NP [O \text{ likes}]][NP [O \text{ likes}][NP [O \text{ likes}][NP [O \text{ likes}][NP [O \text{ likes}]]][NP [O \text{ likes}][NP [O \text{ l
                                                                                                                                                                                                                                                               ?- NP = (findall(X,man(X),L1),findall(V,P2,L2),subset(L1,L2)), VP =
likes(A,john), arg(2,NP,C2), arg(1,C2,FA2), arg(2,FA2,VP), saturate1(FA2,Y),
saturate1(VP,Z).
[S_N = [S_N = S_N = S_
                                                                                                                                                                                                                                                                   S = findall(X,man(X),L1),findall(Y,likes(Y,john),L2),subset(L1,L2)
                                                                                                                                                                                                                                                                   (pass up saturated NP as the value for S)
```

# Names as Generalized Quantifiers

- In earlier lectures, we mentioned that names directly refer
- Here is another idea
- Conjunction
  - X and Y
  - both X and Y have to be of the same type
  - in particular, semantically...
  - we want them to have the same semantic type
- what is the semantic type of every baby?

#### **Example**

```
every baby and John likes ice cream [NP[NP] = VP[NP] =
```

# **Negative Polarity Items**

- Negative Polarity Items (NPIs)
- Examples:
  - every, any
- Constrained distribution:
  - have to be *licensed* in some way
  - grammatical in a "negated environment" or "question"

#### Examples:

- (13a) Shelby won't ever bite you
- (13b) Nobody has any money
- (14a) \*Shelby will ever bite you
- (14b) \*Noah has any money
- \*= ungrammatical
- (15a) Does Shelby ever bite?
- (15b) Does Noah have any money?

# **Negative Polarity Items**

- Inside an if-clause:
  - (16a) If Shelby ever bites you, I'll put him up for adoption
  - (16b) If Noah has any money, he can buy some candy
- Inside an every-NP:
  - (17a) Every dog which has ever bitten a cat feels the admiration of other dogs
  - (17b) Every child who has any money is likely to waste it on candy
- Not inside a some-NP:
  - (17a) Some dog which has ever bitten a cat feels the admiration of other dogs
  - (17b) Some child who has any money is likely to waste it on candy

Not to be confused with free choice (FC) any (meaning: ∀): any man can do that