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**
    - $[_{Sbar}\ [_{S}$ I saw $[_{NP}$ the man $[_{Sbar}$ who $[_{S}$ attacked me]]]]]
  - **possessive NPs**
    - $[_{NP}\ [_{NP}\ [_{NP}$ 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].`

move recursive rule to the end

avoid Infinite Loop in Prolog
ERROR: out of local stack.

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. λx.*
  - *result is a slightly more complicated-looking notation*
- Example:

Phrase	Lambda Calculus	Prolog notation
barks	λx.x barks	`barks(X).`
Shelby barks	[λx.x barks](Shelby)	`barks(X), X = shelby.`

- 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.`

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**
  - *likes* [λy.[λx.x likes y]]
  - *likes Mary* [λy.[λx.x likes y]](Mary)

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

[λy.[λx.x likes y]](Mary)

[λx.x likes **y**] [λy.](**Mary**)

[λx.x likes **Mary**]

More on the Lambda Calculus

Note:

nesting order of λy and λx matters

why:

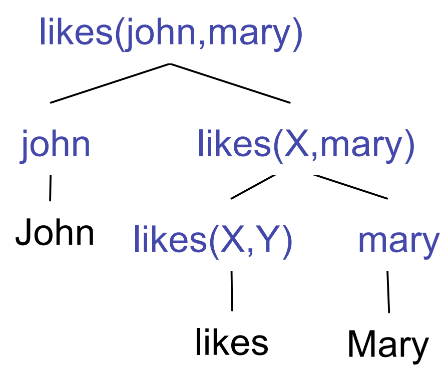
λy.[λx.x likes y]

λx.[λy.x likes y]

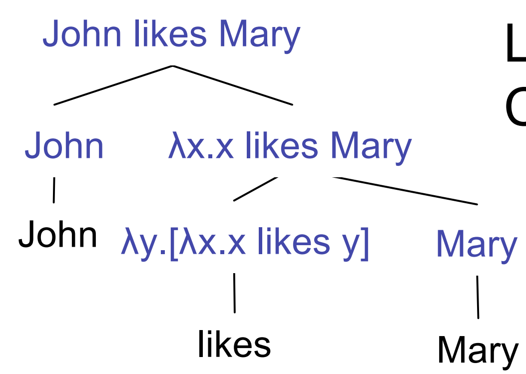
here: lambda expression quantifier for the object must be outside because of phrase structure hierarchy

Example:

Phrase	**Lambda Calculus**
likes	λy.[λx.x likes y]
likes Mary	[λy.[λx.x likes y]](Mary) λx.x likes Mary
John likes Mary	[λx.x likes Mary](John) John likes Mary



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)
    - who_1 Shelby saw e_1 — ***wh*-movement** of $\textit{who}_1$ leaving a trace e_1
    - 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')

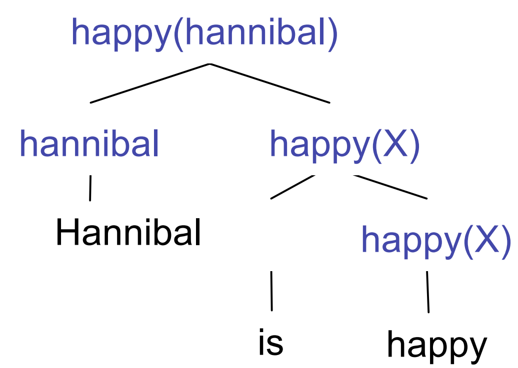
cf. Homework 2
John is a student `student(john).`
John is a baseball fan `baseball_fan(john).`

- In the lambda calculus, the semantics of copula *be* is the **identity function**, e.g. λy.y
- Example Derivation:

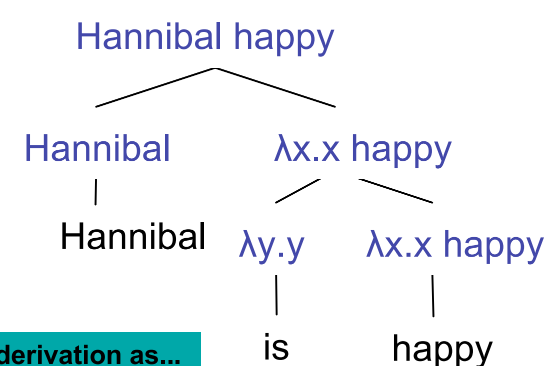
Phrase	Lambda Calculus
is	λy.y
happy	λx.x happy
is happy	[λy.y](λx.x happy)
	λx.x happy

basically the same derivation as...

Phrase	Lambda Calculus
barks	λx.x barks
Shelby barks	[λx.x barks](Shelby)
	Shelby barks



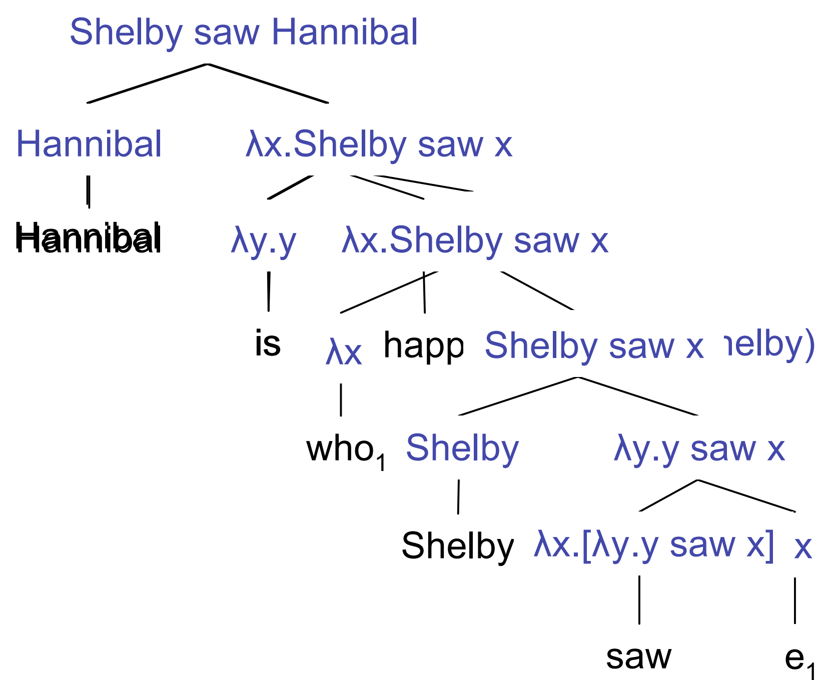
Prolog notation



Lambda calculus

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 who_1 Mary saw e_1
  - (10’) Shelby is [λx.Mary saw x]
  - (10”) Mary saw Shelby