# 6.863J Natural Language Processing
# Lecture 12: Semantics

Robert C. Berwick

## The Menu Bar

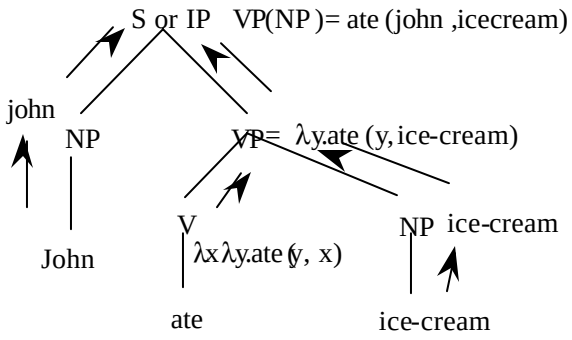
- Administrivia:
  - Schedule alert: Lab 3 due today
  - Lab 4: posted later today – due April 7
- *Agenda:*
- Semantics: why & how
- The great divide: information extraction vs. text understanding

# Example of what we might do: text understanding via q-answering

```
athena>(top-level)
Shall I clear the database? (y or n) y
sem-interpret>John saw Mary in the park
OK.
sem-interpret>Where did John see Mary
IN THE PARK.
sem-interpret>John gave Fido to Mary
OK.
sem-interpret>Who gave John Fido
I DON'T KNOW
sem-interpret>Who gave Mary Fido
JOHN
sem-interpret >John saw Fido
OK.
sem-interpret>Who did John see
FIDO AND MARY
```



# How: recover meaning from structure



# "Logical" semantic interpretation

- Four basic principles

1. **Rule-to-Rule** semantic interpretation [aka *"syntax-directed translation"*]: pair syntax, semantic rules. (GPSG: pair each cf rule w/ semantic 'action'; as in compiler theory – due to Knuth, 1968)
2. **Compositionality:** Meaning of a phrase is a function of the meaning of its parts *and nothing more* e.g., meaning of S→NP VP is *f*(M(NP)• M(VP)) (analog of 'context-freeness' for semantics – local)
3. **Truth conditional meaning**: meaning of S equated with *conditions* that make it true
4. **Model theoretic semantics**: correlation betw. Language & world via set theory & mappings



# Answer 1: translation – from 'syntactic' rep to 'semantic' rep, aka "Deep"

- Mirrors the progamming language approach
- When is it used?
- DB Q&A (but answer 2 can be used here...when and how?)
- Text understanding: when *all* the text is relevant - voice, inference, paraphrase, important
- Intentions, beliefs, desires (non-extensional= not just sets of items)

# Answer 2 – 'Shallow' – information extraction

- What do we need to know to get this task done?
- Slot-and-filler semantics
- Limited parsing, limited predicate-arguments
- Let's see what we need to know about 'meaning' by looking at an example



# Example – news stories/MUC

**Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and "metal wood" clubs a month.**

**TIE-UP-1:**
**Relationship: TIE-UP**
**Entities: "Bridgestone Sports Co."**
**"a local concern"**
**"a Japanese trading house"**
**Joint Venture Company: "Bridgestone Sports Taiwan Co."**
**Activity: ACTIVITY-1**
**Amount: NT$20000000**

**ACTIVITY-1:**
**Activity: PRODUCTION**
**Company: "Bridgestone Sports Taiwan Co."**
**Product: "iron and `metal wood' clubs"**
**Start Date: DURING: January 1990**

## Vs. this task...

Person: Put the blue block on the pyramid

System: I'm going to have to clear off the pyramid. Oops, I can't do that – a pyramid can't support the block.

OK, move it onto the red block.

OK.

What supports the blue block?

The red block.



## Key questions

- What do we have to *know* in order to get the job done?
- And then – how do we *represent* this knowledge?
- And then – how do we *compute* with this representation?
- (cf. David Marr's notions)

## Answers defined in terms of characteristics of 'the task'

- Information extraction
  - Function is communication of factual information
  - Typically only parts of the text are relevant
  - Typically only part of a relevant sentence is relevant
  - Only predicate-argument structure needed (at a superficial level)
  - No modeling of author or audience



## 'Shallow' or IE task

- Predicate-arguments: 'who did what to whom' – in fact, just a core set of verbs that are relevant (e.g., if business merger, 'set up', 'produce',... etc.)
- Extract simple relationships among singular entities
- E.g., `X set up a joint-venture with Y'

# Example – news stories/MUC

**Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and "metal wood" clubs a month.**

**TIE-UP-1:**
**Relationship: TIE-UP**
**Entities: "Bridgestone Sports Co."**
**"a local concern"**
**"a Japanese trading house"**
**Joint Venture Company: "Bridgestone Sports Taiwan Co."**
**Activity: ACTIVITY-1**
**Amount: NT$20000000**

**ACTIVITY-1:**
**Activity: PRODUCTION**
**Company: "Bridgestone Sports Taiwan Co."**
**Product: "iron and `metal wood' clubs"**
**Start Date: DURING: January 1990**



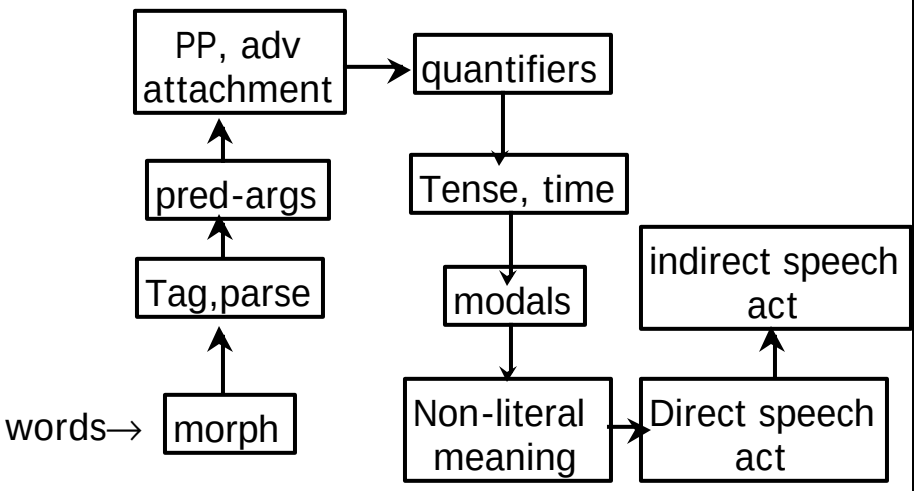
# Even the parsing is shallow

- Chunking – no recursion, just p.o.s brackets
- **[Bridgestone Sports Co.][**said **[Friday] ] [it ] has [ set up [ a joint venture ] ] in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan**

Can use simple linear patterns:

chunk: NP -> D? N+;
VP -> V-tns | Aux V-ing
clause: S -> PP* NP PP* VP PP*

# What we give up – in terms of 'big picture'






# IE or Shallow

- Parsing:
  - PP attachment – ignored, except for arguments of domain relevant verbs
    "set up a joint venture" vs. "a joint venture in Japan" vs. "a joint venture in their home office"
  - Adverbials – only locatives, temporal adverbs; others ignored (why?)
- Semantics:
  - No modals (might, will, could…)
  - No propositional attitudes, possible worlds, user intentions, etc. (believe, want, unicorns,…)
  - Non-literal meaning

## What's all this stuff that's added?

- Parsing
  - Details of all phrase attachments - exact
- Logical Semantic additions:
  - All arguments to all predicate-argument structure
  - Adjunct modifiers
  - Quantifiers
  - Detailed, accurate tense representation
  - Modal verbs
  - Propositional attitudes, belief contexts
  - Direct and indirect speech acts



## What's all this stuff??

- Quantifiers
  - John ate an ice-cream
  - Ice-cream now not a constant
  - John ate an ice-cream and Mary ate an ice-cream
- In the set of ice-creams, there exists one eaten by John
- Ice-cream a predicate on entities
- Can compute using sets (extensional)

# What's all this stuff?

- Tense
  - "There was an event some time in the past such that an ice-cream was among the objects eaten by John at that time"
  - Could just use a variable t
  - We will improve this representation later
- Why stop there? Events have other properties



# This gets complex

- John ate an ice-cream in a booth
  - Event representation
  - ∃e past(e), act(e,eating), eater(e,John), exists(ice-cream, eatee(e)), exists(booth, location(e))
- John ate an ice-cream in every booth
  - ∃e past(e), act(e,eating), eater(e,John), exists(ice-cream, eatee(e)), all(booth, location(e)),

∃g ice-cream(g), eatee(e,g)  ∀b booth(b)⇒location(e,b)

## So this means..

- This means ∃e ∀b which means same event for every booth
- False unless John can be in every booth during his eating of a single ice-cream
- Which order do we want?
- ∃b ∀e: “for all booths b, there was such an event in b”

- Figuring this out requires a notion of scope (and so, structure…)
- But wait, there’s more… what about *all, none, …*



## Beliefs, Desires and Intentions

- How do we represent internal speaker states like believing, knowing, wanting, assuming, imagining..?
  - Not well modeled by a simple DB lookup approach
  - Truth in the world vs. truth in some possible world

  George imagined that he could dance.

  Geroge believed that he could dance.
- Augment FOPC with special modal operators that take logical formulae as arguments, e.g. believe, know

# Intensional Arguments

- John wants a unicorn (cf., John wants an ice-cream)
  - "there is a unicorn u that Willy wants"
  - here the wantee is an individual entity
  - "Willy wants any entity u that satisfies the unicorn predicate"
  - here the wantee is a type of entity
- Problem
  - 'unicorn' is defined by the set of unicorns – its extension
  - BUT this set is empty
  - All empty sets are equal (but some are more equal than others...)
  - So, John wants a unicorn ≡ John wants a dodo
  - What's wanted (wantee) should be intension or criteria for being a unicorn
- (One) solution: possible world semantics:
  - Can imagine other worlds where set of unicorn $^1$ set of dodos



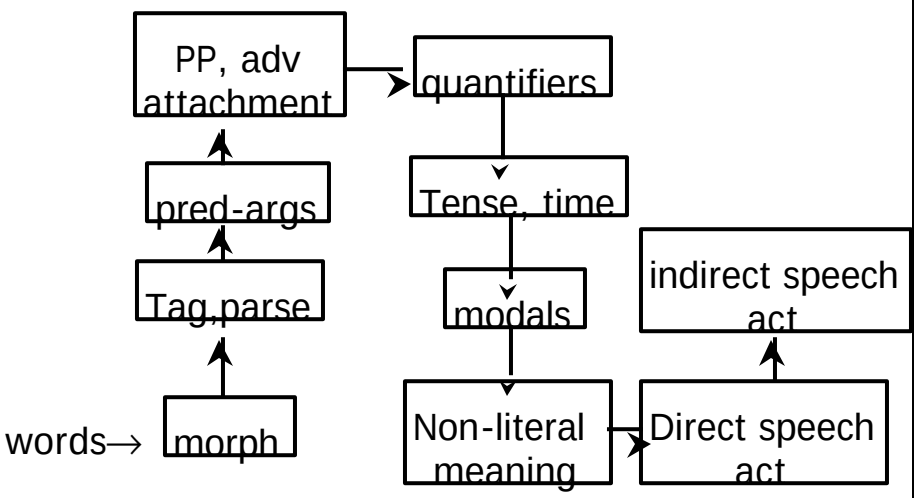
- Mutual belief: I believe you believe I believe....
  - Practical importance: modeling belief in dialogue

# Non-literal meaning (source of 60% of old Star Trek plots)

- Kirk: Spock, are there any Romulans in Sector 6471?
- Spock: None, captain.
- Kirk: Are you certain, Spock?
- Spock: A 100% probability, Captain
- [camera rolls] Kirk: Damn your Vulcan ears, Spock, I thought you said there were no Vulcans in sector 6471!!&*(!&
- Spock: But there is no sector 6471...Logic dictates... [Fadeout to commercial]



# What we give up – in terms of 'big picture'



## Illustrations – indirect speech act

- It's cold in here
- What would 'shallow approach' do?
- What about 'full understanding' – indirect speech act
- What about discourse:
- Guiliani left Bloomberg to be mayor of a city with a big budget problem. It's unclear how he'll be able to handle it during his term.



## Why lunch at Lobdell is slow

"Do you have the salt" $\rightarrow$ "Please pass the salt"

# Some complications

- Temporal logic
  - Gilly had swallowed eight goldfish before Milly reached the bowl
  - Billy said my pet fish was pregnant
  - Billy said, "my pet fish is pregnant."
- Generics
  - Typhoons arise in the Pacific
  - Children must be carried
- Presuppositions
  - The king of France is bald.
- Pronoun-Quantifier Interaction ("bound anaphora")
  - Every farmer who owns a donkey beats it.
  - If you have a dime, put it in the meter.
  - The woman who every Englishman loves is his mother.
  - I love my mother and so does Billy.



# Classical (logical) semantic interpretation

- Four basic principles

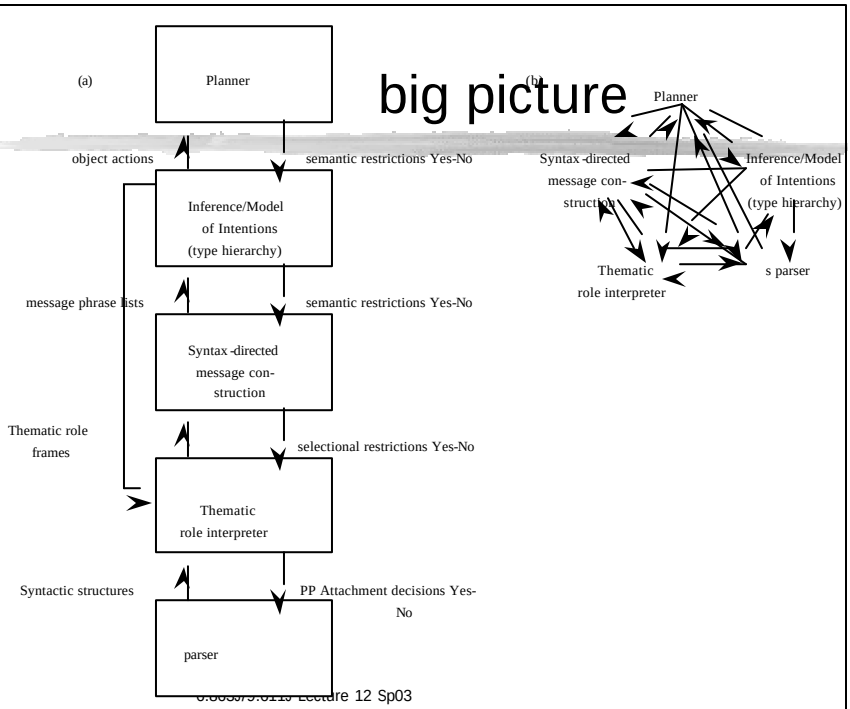
1. **Rule-to-Rule** semantic interpretation [aka *"syntax-directed translation"*]: pair syntax, semantic rules. (GPSG: pair each cf rule w/ semantic 'action'; as in compiler theory – due to Knuth, 1968)
2. **Compositionality:** Meaning of a phrase is a function of the meaning of its parts *and nothing more* e.g., meaning of S→NP VP is *f*(M(NP)• M(VP)) (analog of 'context-freeness' for semantics – local)
3. **Truth conditional meaning**: meaning of S equated with *conditions* that make it true
4. **Model theoretic semantics**: correlation betw. Language & world via set theory & mappings (extensional)

# Components

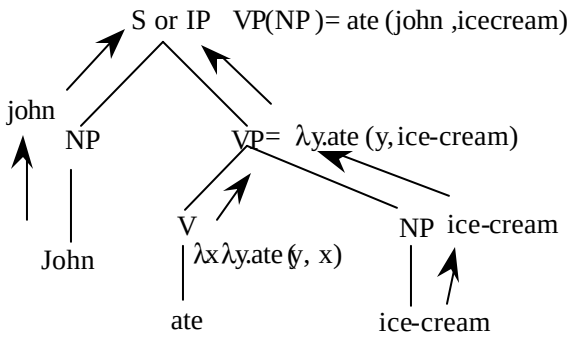
- Semantic representation (“logical form”)
  - Start w/ lambda calculus, predicates
  - Patch as we encounter phenomena in language: quantifiers,



# big picture



# How: recover meaning from structure






# What Counts as Understanding?
## some notions

- We understand if we can respond appropriately
  - ok for commands, questions (these demand response)
  - "Computer, warp speed 5"
  - "throw axe at dwarf"
  - "put all of my blocks in the red box"
  - imperative programming languages
  - database queries and other questions
- We understand statement if we can determine its truth
  - ok, but if you knew whether it was true, why did anyone bother telling it to you?
  - comparable notion for understanding NP is to compute what the NP refers to, which might be useful

# Representing Meaning

- What requirements do we have for meaning representations?



# What requirements must meaning representations fulfill?

- Verifiability: The system should allow us to compare representations to facts in a Knowledge Base (KB)
  - Cat(Huey)
- Ambiguity: The system should allow us to represent meanings unambiguously
  - German teachers has 2 representations
- Vagueness: The system should allow us to represent vagueness
  - He lives somewhere in the south of France.

## Requirements: Inference

- Draw valid conclusions based on the meaning representation of inputs and its store of background knowledge.

  Does Huey eat kibble?

  thing(kibble)

  Eat(Huey,x) ^ thing(x)



## What Counts as Understanding?

- Be able to translate
  - Depends on target language
  - English to English? bah humbug!
  - English to French? reasonable
  - English to Chinese? requires deeper understanding
  - English to *logic*? deepest

  all humans are mortal = $\forall x$ [human(x) $\Rightarrow$mortal(x)]

- Assume we have logic-manipulating rules to tell us how to act, draw conclusions, answer questions ...

# Requirements: Canonical Form

- Inputs that mean the same thing have the same representation.
  - Huey eats kibble.
  - Kibble, Huey will eat.
  - What Huey eats is kibble.
  - It's kibble that Huey eats.
- Alternatives
  - Four different semantic representations
  - Store all possible meaning representations in Knowledge Base



# Requirements: Compositionality

- Can get meaning of "brown cow" from *separate, independent* meanings of "brown" and "cow"
- Brown(x)$\wedge$ Cow(x)
- I've never seen a purple cow, I never hope to see one...

## Barriers to compositionality

- *Ce corps qui s'appelait e qui s'appelle encore le saint empire romain n'etait en aucune maniere ni saint, ni romain, ni empire.*
- This body, which called itself and still calls itself the Holy Roman Empire, was neither Holy, nor Roman, nor an Empire -*Voltaire*



## Need some kind of logical calculus

- Not ideal as a meaning representation and doesn't do everything we want - but close
  - Supports the determination of truth
  - Supports compositionality of meaning
  - Supports question-answering (via variables)
  - Supports inference
- What are its elements?
- What else do we need?

# The elements

Three major kinds of objects

1. Booleans
   - Roughly, the semantic values of sentences
2. Entities
   - Values of NPs, i.e., objects
   - Maybe also other types of entities, like times
3. Functions of various types
   - A function returning a boolean is called a "predicate" – e.g., frog(x), green(x)
   - Functions might return other functions!
   - Function might take other functions as arguments!



# Syntax for this calculus

- Terms: constants, functions, variables
  - Constants: objects in the world, e.g. Huey
  - Functions: concepts, e.g. sisterof(Huey)
  - Variables: x, e.g. sisterof(x)
- Predicates: symbols that refer to relations that hold among objects in some domain or properties that hold of some object in a domain

  likes(Huey, kibble)

  cat(Huey)

- Logical connectives permit compositionality of meaning

  kibble(x) $\rightarrow$ likes(Huey,x)

  cat(Vera) ^ weird(Vera)

  sleeping(Huey) v eating(Huey)
- Expressions can be assigned truth values, T or F, based on whether the propositions they represent are T or F in the world
  - Atomic formulae are T or F based on their presence or absence in a DB (Closed World Assumption?)
  - Composed meanings are inferred from DB and meaning of logical connectives



  - cat(Huey)
  - sibling(Huey,Vera)
  - sibling(x,y) ^ cat(x) $\rightarrow$ cat(y)
  - cat(Vera)??
- Limitations:
  - Do 'and' and 'or' in natural language really mean '^' and 'v'?

    Mary got married and had a baby.

    Your money or your life!

    He was happy but ignorant.
  - Does '$\rightarrow$' mean 'if'?

    I'll go if you promise to wear a tutu.

# What Can Serve as a Meaning Representation?

- Anything that serves the core practical purposes of a program that is doing semantic processing …
  - Answer questions (What is the tallest building in the world?)
  - Determining truth (Is the blue block on the red block?)
  - Drawing inferences (If the blue block is on the red block and the red block is on the tallest building in the world, then the blue block is on the tallest building in the world)

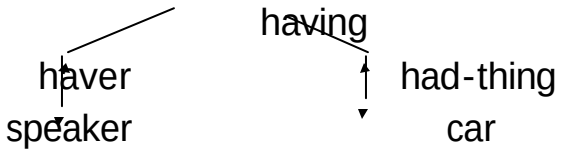


# Common Meaning Representations

- First order predicate calculus (FOPC):

$$\exists x, y Having(x) \wedge Haver(S, x) \wedge HadThing(y, x) \wedge Car(y)$$

- Semantic Net:




- Conceptual Dependency Diagram:

Car
⇑ Poss-By
Speaker

- Frame
  - Having
    - Haver: S
    - HadThing: Car
- All represent ‘linguistic meaning’ of I have a car

*and* state of affairs in some world

- All consist of structures, composed of symbols representing objects and relations among them



## Expressiveness

- Must accommodate wide variety of meanings

# Predicate-Argument Structure

- Represents concepts and relationships among them
  - Nouns as concepts or arguments (red(ball))
  - Adjectives, adverbs, verbs as predicates (red(ball))
- Subcategorization (or, argument) frames specify number, position, and syntactic category of arguments
  - NP likes NP
  - NP likes [to eat ice-cream]



# Thematic Roles

- Subcat frames link arguments in surface structure with their semantic roles
  - Agent: George hit Bill. Bill was hit by George.
  - Patient: George hit Bill. Bill was hit by George.
- Selectional Restrictions: constraints on the ***types***

of arguments verbs take

***George assassinated the senator.***

****The spider assassinated the fly.***

***assassinate: intentional (political?) killing***

## What

- What representation do we want for something like

  John ate ice-cream $\rightarrow$

  ate(John, ice-cream)
- Lambda calculus
- We'll have to posit something that will do the work
- Predicate of 2 arguments:

  $\lambda x \lambda y$ ate(y, x)



## What: Basic semantic representation: or 'thematic role' frame

- Use of "Event structure" (recursive)

  **`(EVENT :condition1 val1 :condition2 val2…`**
  **`:condn valn)`**

Example:

- (see :agent John :patient Mary :tense past)
- Sometimes called a 'thematic role frame' or (earlier): 'case frame' (Fillmore, 1965)

# More complex example

(cause :agent (bob) :effect (go :theme (book) :path
(path :oper (onto) :terminal+ (shelf))) :tense
past)



# Meaning of sentence

- Is the application of the lambda form associated with the VP to the lambda form given by the argument NP
- Words just return 'themselves' as values (from lexicon)
- Given parse tree, then by working bottom up as shown next, we get to the logical form *ate(John, ice-cream)*
- This predicate can then be evaluated against a database – this is *model interpretation*- to return a value, or t/f, etc.

# Lambda application works

- Suppose John, ice-cream = constants, i.e., $\lambda x.x$, the identity function
- Then lambda substitution does give the right results:

  $\lambda x\, \lambda y$ ate(y, x) (ice-cream)(John)$\rightarrow$

  $\lambda y$ ate(y, ice-cream)(John)$\rightarrow$

  ate(John, ice-cream)

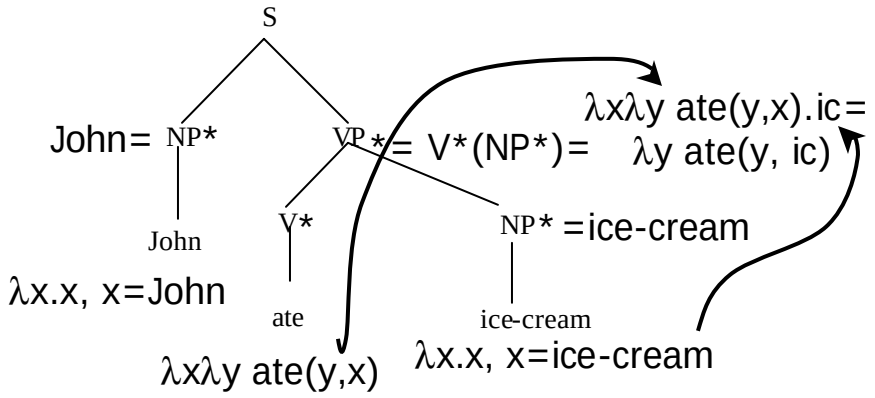
But... where do we get the $\lambda$–forms from?



# Example of what we now can do

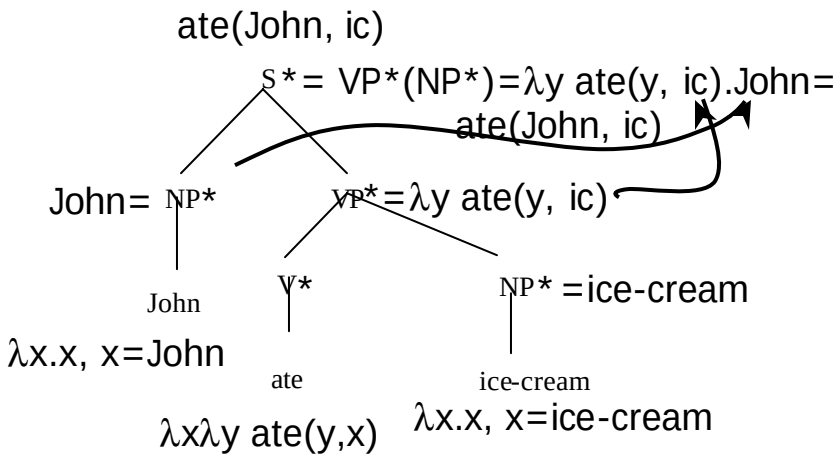
```
athena>(top-level)
Shall I clear the database? (y or n) y
sem-interpret>John saw Mary in the park
OK.
sem-interpret>Where did John see Mary
IN THE PARK.
sem-interpret>John gave Fido to Mary
OK.
sem-interpret>Who gave John Fido
I DON'T KNOW
sem-interpret>Who gave Mary Fido
JOHN
sem-interpret >John saw Fido
OK.
sem-interpret>Who did John see
FIDO AND MARY
```

# How: to recover meaning from structure






# How



## In this picture

- The meaning of a sentence is the composition of a function VP* on an argument NP*
- The lexical entries are λ forms
  - Simple nouns are just constants
  - Verbs are λ forms indicating their argument structure
- Verb phrases return λ functions as their results (in fact – higher order)



## How

- Application of the lambda form associated with the VP to the lambda form given by the argument NP
- Words just return 'themselves' as values (from lexicon)
- Given parse tree, then by working bottom up as shown next, we get to the logical form *ate(John, ice-cream)*
- This predicate can then be evaluated against a database – this is *model interpretation*- to return a value, or t/f, etc.

# Code – sample rules

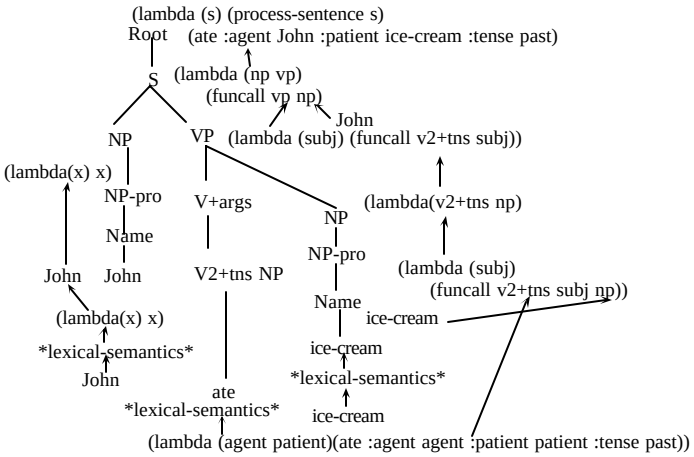
*Syntactic rule* *Semantic rule*

```
(root ==> s)          (lambda (s)(PROCESS-SENTENCE s))

(s ==> np vp)         (lambda (np vp)(funcall vp np)))

(vp ==> v+args)       (lambda (v+args)(lambda (subj)
                          (funcall v+args subj))))

(v+args ==> v2 np)(lambda (v2 np)
                        (lambda (subj)
                            (funcall v2 subj np))))

(v kiss)              (lambda (agent beneficiary affcted-obj))

(np-pro ==> name) #'identity)
```

**Verb arguments**



# The semantic interpreter procedure



# How does this work?

- Top level lambda says to call procedure named VP (whose value will be determined "from below", ie, S-I of VP) by using the *arg* NP (again whose meaning will be provided "from below)
- In other words, to find the meaning of S, we call the procedure VP using as an argument the subject NP
- These two values will be supplied by the (recursive) semantic interpretation of the NP and VP nodes.
- At the very bottom, individual words must also contain some paired 'semantic' value
- This is almost enough to do the code for the whole example!



# Code – sample rules

*Syntactic rule*

*Semantic rule*

```
add-rule-semantics '(root ==> s)
                   '(lambda (s)
                      (PROCESS-SENTENCE s)))

(add-rule-semantics '(s ==> np vp)
                    #'(lambda (np vp)
                        (funcall vp np)))

(add-rule-semantics '(vp ==> v+args)
                #'(lambda (v+args)
                  #'(lambda (subj)
                      (funcall v+args subj))))

(add-rule-semantics '(v+args ==> v2 np)
                    #'(lambda (v2 np)
                       #'(lambda (subj)
                            (funcall v2 subj np))))

(add-rule-sem '(np-pro ==> name) #'identity)
```

# Code – the interpreter

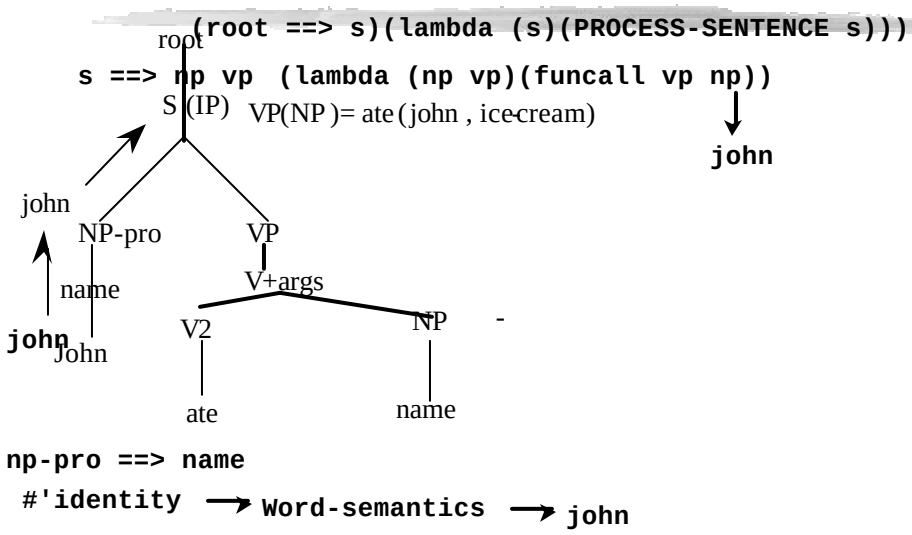
```
;;Parse rules into syntactic/semantic parts, recursively
(defun phrase-semantics (phrase)
  (cond ((atom (second phrase))  ; find phrase name –a word?
         (word-semantics (second phrase) (first phrase))) ; o.w.
        (t (rule-apply (rule-semantics (first phrase)   ; recurse
                                       (mapcar
                                              #'first(rest phrase)))
                                       (mapcar #'phrase-semantics
                                              (rest phrase))))))

;; now apply-eval loop for the semantic rules
(defun rule-apply (head args)
  (let ((result (apply head args)))
    (if (and (consp result)
             (eq (first result) 'lambda))
        (eval (list 'function result))
        result)))
```



# Code for this

```
(defun word-semantics (word sense)
  (let ((x (lookup2 word sense *lexical-semantics*)))
    (if (and (consp x)
             (eq (first x) 'lambda))
        (eval (list 'function x))
        x)))

(defun rule-semantics (head args)
  (let ((x (lookup2 head args *phrasal-semantics*)))
    (if (and (consp x)
             (eq (first x) 'lambda))
        (eval (list 'function x))
        x)))
```

# Construction step by step – on NP side






# In this picture

- The meaning of a sentence is the composition of a function VP* on an argument NP*
- The lexical entries are λ forms
  - Simple nouns are just constants
  - Verbs are λ forms indicating their argument structure
- Verb phrases return a function as its result

# Syntax & paired semantics

| Item or rule | Semantic translation |
|---|---|
| Verb *ate* | $\lambda x \lambda y.ate(y, x)$ |
| propN | $\lambda x.x$ |
| V | V* = $\lambda$ for lex entry |
| S (or CP) | S* = VP*(NP*) |
| NP | N* |
| VP | V*(NP*) |



# Logic: Lambda Terms

- Lambda terms:
  - A way of writing "anonymous functions"
    - No function header or function name
    - But defines the key thing: **behavior** of the function
    - Just as we can talk about 3 without naming it "x"
  - Let square = $\lambda$p p*p
  - Equivalent to int square(p) { return p*p; }
  - But we can talk about $\lambda$p p*p without naming it
  - Format of a lambda term: $\lambda$ variable expression

# Logic: Lambda Terms

- Lambda terms:
  - Let square = λp p*p
  - Then square(3) = (λp p*p)(3) = 3*3
  - Note: square(x) isn't a function! It's just the value x*x.
  - But l**x** square(x) = λx x*x = λp p*p = square
    (proving that these functions are equal – and indeed they are, as they act the same on all arguments: what is (λx square(x))(y)? )

---

  - Let even = λp (p mod 2 == 0) a predicate: returns true/false
  - even(x) is true if x is even
  - How about even(square(x))?
  - λx even(square(x)) is true of numbers with even squares
    - Just apply rules to get λx (even(x*x)) = λx (x*x mod 2 == 0)
    - This happens to denote the same predicate as even does



# Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...

- Suppose we want to write times(5,6)
- Remember: square can be written as λx square(x)
- Similarly, times is equivalent to λx λy times(x,y)

- Claim that times(5)(6) means same as times(5,6)
  - times(5) = (λx λy times(x,y)) (5) = λy times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (λy times(5,y))(6) = times(5,6)

# Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...
- Claim that times(5)(6) means same as times(5,6)
  - times(5) = ($\lambda$x $\lambda$y times(x,y)) (5) = $\lambda$y times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = ($\lambda$y times(5,y))(6) = times(5,6)
- So we can always get away with 1-arg functions ...
  - ... which might return a function to take the next argument. Whoa.
  - We'll still allow times(x,y) as syntactic sugar, though



# Grounding out

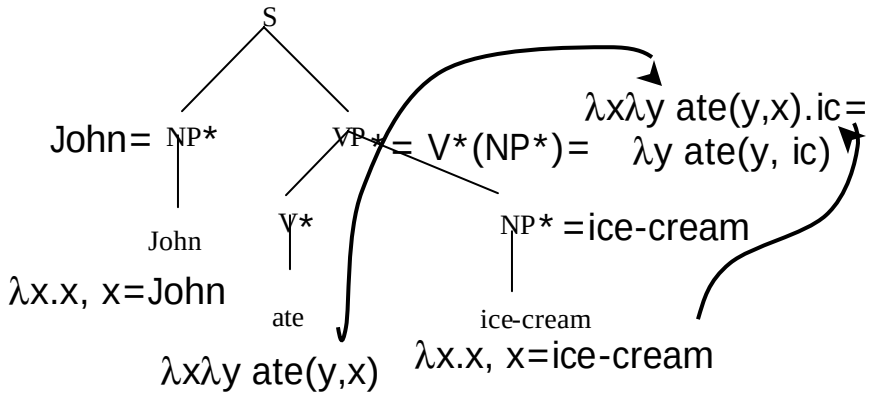
- So what does times actually mean???
- How do we get from times(5,6) to 30 ?
  - Whether times(5,6) = 30 depends on whether symbol times actually denotes the multiplication function!
- Well, maybe times was defined as another lambda term, so substitute to get times(5,6) = (blah blah blah)(5)(6)
- But we can't keep doing substitutions forever!
  - Eventually we have to ground out in a primitive term
  - Primitive terms are bound to object code
- Maybe times(5,6) just executes a multiplication function
- What is executed by loves(john, mary) ?

## Logic: Interesting Constants

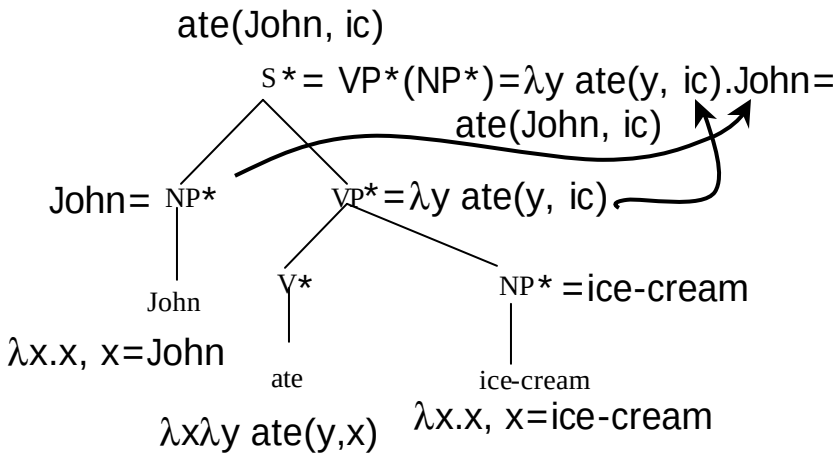
- Thus, have "constants" that name some of the entities and functions (e.g., times):
  - Eminem - an entity
  - red – a predicate on entities
    - holds of just the red entities: red(x) is true if x is red!
  - loves – a predicate on 2 entities
    - loves(Eminem,Detroit)
    - *Question:* What does loves(Detroit) denote?
- Constants used to define meanings of words
- Meanings of phrases will be built from the constants & syntactic structure



## How: to recover meaning from structure



# How






# Processing options

- Off-line vs. on-line
- Off-line: do all syntax first, then pass to semantic interpretation (via pass on syntax tree(s))
- On-line: do it as each phrase is completed

# On-line

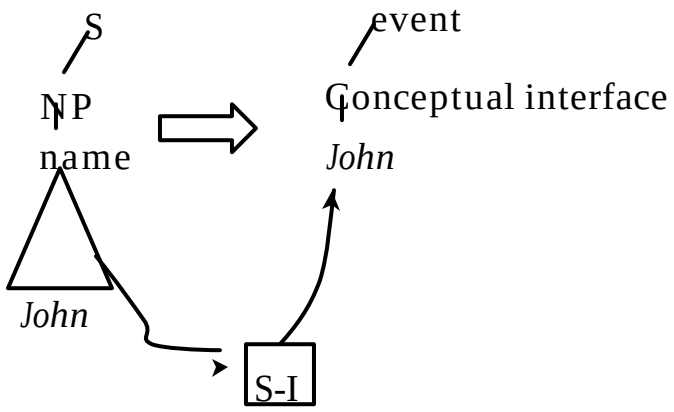
S → NP VP {VP*(NP*)}

- VP* has been stored in state representing VP
- NP* stored with the state for NP
- When rule completed, go get value of VP*, go get NP*, and apply VP* to NP*
- Store result in S*.

- As fragments of input parsed, semantic fragments created
- Can be used to block ambiguous representations



# Picture



## Processing order: online

- *Interpret* subtree as soon as it is built –eg, as soon as RHS of rule is finished (complete subtree)
- Picture: "ship off" subtree to semantic interpretation as soon as it is "done" syntactically
- Allows for off-loading of syntactic short term memory; SI returns with 'ptr' to the interpretation
- Natural order to doing things (if process left to right)
- Has some psychological validity – tendency to interpret asap & lower syntactic load
- Example: *I told John a ghost story vs. I told John a ghost story was the last thing I wanted to hear*



## Drawback

- You also perform semantic analysis on orphaned constituents that play no role in final parse
- Worst case:
- Jump out the window,
  - But not before you put on your parachute
- Hence, case for pipelined approach: Do semantics *after* syntactic parse

# Doing Compositional Semantics

- To incorporate semantics into grammar we must
  - Figure out right representation for a single constituent based on the parts of that constituent (e.g. Adj)
  - Figuring out the right representation for a category of constituents based on other grammar rules making use of that constituent (e.g NP→ Adj Noun)
- This gives us a set of function-like semantic attachments incorporated into our CFG
  - E.g. NP → Adj Noun* {λx Noun*(x) ^ Isa(x,Adj*)}



# What do we do with them?

- As we did with feature structures:
  - Alter an Early-style parser so when constituents (dot at the end of the rule) are completed, the attached semantic function applied and meaning representation created and stored with state
- Or, let parser run to completion and then walk through resulting tree running semantic attachments from bottom-up

## What can we do with this machinery?

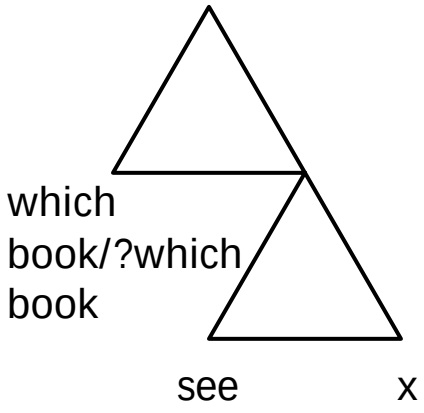
- A lot (almost all): start adding phenomena (figure out the representation) – and see
- To begin: wh-moved NPs (which book...), which act *just like* other quantifiers



## Wh questions

- Part of process-sentence
- Wh form is placed by semantics in template as, eg, ?which or ?who
- This will then correspond to the "for which x, x a person" typed lambda calculus form we wanted – explicitly in a procedural way
- Procedure prompts a *search* through db for matching sets of items that can align w/ the template

# Picture – wh-NP & trace exactly in correct configuration






# Summing Up

- Hypothesis: Principle of Compositionality
  - Semantics of NL sentences and phrases can be composed from the semantics of their subparts
- Rules can be derived which map syntactic analysis to semantic representation (Rule-to-Rule Hypothesis)
  - Lambda notation provides a way to extend FOPC to this end
  - But coming up with rule2rule mappings is hard
- Idioms, metaphors perplex the process