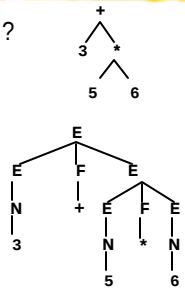
Semantics

From Syntax to Meaning!

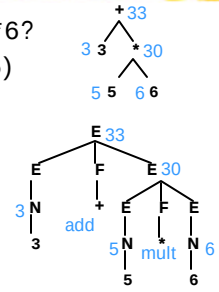
Programming Language Interpreter

- What is meaning of 3+5*6?
- First parse it into 3+(5*6)



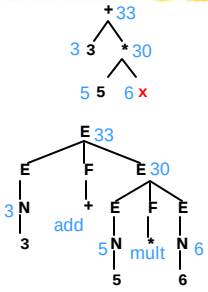
Programming Language Interpreter

- What is meaning of 3+5*6?
- First parse it into 3+(5*6)
- Now give a meaning to each node in the tree (bottom-up)



Interpreting in an Environment

- How about 3+5*x?
- Same thing: the meaning of x is found from the environment (it's 6)
- Analogies in language?

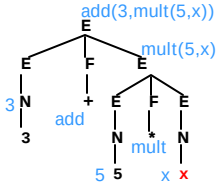


Compiling

- How about 3+5*x?
- Don't know x at compile time
- "Meaning" at a node is a piece of code, not a number

`5*(x+1)-2` is a different expression that produces *equivalent* code (can be converted to the previous code by optimization)

Analogies in language?



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

What Counts as Understanding? some notions

- We understand statement if we know *how* to determine its truth
  - What are exact conditions under which it would be true?
    - necessary + sufficient
  - Equivalently, derive all its consequences
    - what else must be true if we accept the statement?
  - Philosophers tend to use this definition
- We understand statement if we can use it to answer questions [very similar to above – requires reasoning]
  - **Easy:** John ate pizza. What was eaten by John?
  - **Hard:** White's first move is P-Q4. Can Black checkmate?
  - Constructing a *procedure* to get the answer is enough

What Counts as Understanding? some notions

- 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 - the definition we'll use!
    - all humans are mortal = ?x [human(x) ? mortal(x)]
- Assume we have logic-manipulating rules to tell us how to act, draw conclusions, answer questions ...

Lecture Plan

- Today:
  - Let's look at some sentences and phrases
  - What would be reasonable logical representations for them?
- Tomorrow:
  - How can we build those representations?
- Another course (AI):
  - How can we reason with those representations?

Logic: Some Preliminaries

Three major kinds of objects

1. Booleans
   - Roughly, the semantic values of sentences
2. Entities
   - Values of NPs, e.g., objects like this slide
   - 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!

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 = ?p p*p
  - Equivalent to int square(p) { return p*p; }
  - But we can talk about ?p p*p without naming it
  - Format of a lambda term: ? variable expression

Logic: Lambda Terms

- Lambda terms:
  - Let square = ?p p*p
  - Then square(3) = (?p p*p)(3) = 3*3
  - Note: square(x) isn't a function! It's just the value x*x.
  - But ?**x** square(x) = ?x x*x = ?p p*p = square

    (proving that these functions are equal – and indeed they are, as they act the same on all arguments: what is (?x square(x))(y)?)

  - Let even = ?p (p mod 2 == 0) a predicate: returns true/false
  - even(x) is true if x is even
  - How about even(square(x))?
  - ?x even(square(x)) is true of numbers with even squares
    - Just apply rules to get ?x (even(x*x)) = ?x (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 ?x square(x)
- Similarly, times is equivalent to ?x ?y times(x,y)

- Claim that times(5)(6) means same as times(5,6)
  - times(5) = (?x ?y times(x,y)) (5) = ?y times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (?y 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) = (?x ?y times(x,y)) (5) = ?y times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (?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):
  - GeorgeWBush - 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(GeorgeWBush, LauraBush)
    - *Question:* What does loves(LauraBush) denote?
- Constants used to define meanings of words
- Meanings of phrases will be built from the constants

Logic: Interesting Constants

- most – a predicate on 2 predicates on entities
  - most(pig, big) = "most pigs are big"
    - Equivalently, most(?x pig(x), ?x big(x))
  - returns true if most of the things satisfying the first predicate also satisfy the second predicate
- similarly for other quantifiers
  - all(pig,big) (equivalent to ? x pig(x) ? big(x))
  - exists(pig,big) (equivalent to ?x pig(x) AND big(x))
  - can even build complex quantifiers from English phrases:
    - "between 12 and 75"; "a majority of "; "all but the smallest 2"

A reasonable representation?

- `Gilly swallowed a goldfish`
- First attempt: swallowed(Gilly, goldfish)

- Returns true or false. Analogous to
  - prime(17)
  - equal(4,2+2)
  - loves(GeorgeWBush, LauraBush)
  - swallowed(Gilly, Jilly)
- … or is it analogous?

A reasonable representation?

- `Gilly swallowed a goldfish`
  - First attempt: swallowed(Gilly , goldfish)
- But we're not paying attention to a!
- `goldfish` isn't the name of a unique object the way `Gilly` is
- In particular, don't want
  `Gilly swallowed a goldfish and Milly swallowed a goldfish`
  to translate as
  swallowed(Gilly , goldfish) AND swallowed(Milly , goldfish)
  since probably not the same goldfish ...

Use a Quantifier

- `Gilly swallowed a goldfish`
  - First attempt: swallowed(Gilly , goldfish)
- Better: ?g goldfish(g) AND swallowed(Gilly , g)
- Or using one of our quantifier predicates:
  - exists(?g goldfish(g), ?g swallowed(Gilly,g))
  - Equivalently: exists(goldfish, swallowed(Gilly))
    - "In the set of goldfish there exists one swallowed by Gilly"
- Here goldfish is a predicate on entities
  - This is the same semantic type as red
  - But `goldfish` is noun and `red` is adjective .. #@!?

Tense

- `Gilly swallowed a goldfish`
  - Previous attempt: exists(goldfish, ?g swallowed(Gilly,g))
- Improve to use tense:
  - Instead of the 2-arg predicate swallowed(Gilly,g)
    try a 3-arg version swallow(t,Gilly,g) where t is a time
  - Now we can write:
    ?t past(t) AND exists(goldfish, ?g swallow(t,Gilly,g))
  - "There was some time in the past such that a goldfish was among the objects swallowed by Gilly at that time"

(Simplify Notation)

- `Gilly swallowed a goldfish`
  - Previous attempt: exists(goldfish, swallowed(Gilly))
- Improve to use tense:
  - Instead of the 2-arg predicate swallowed(Gilly,g)
    try a 3-arg version swallow(t,Gilly,g)
  - Now we can write:
    ?t past(t) AND exists(goldfish, swallow(t,Gilly))
  - "There was some time in the past such that a goldfish was among the objects swallowed by Gilly at that time"

Event Properties

- `Gilly swallowed a goldfish`
  - Previous: ?t past(t) AND exists(goldfish, swallow(t,Gilly))
- Why stop at time? An event has other properties:
  - `[Gilly] swallowed [a goldfish] [on a dare] [in a telephone booth] [with 30 other freshmen] [after many bottles of vodka had been consumed].`
  - Specifies who what why when ...
- Replace time variable t with an event variable e
  - ?e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
    - As with probability notation, a comma represents AND
    - Could define past as ?e ?t before(t,now), ended-at(e,t)

Quantifier Order

- `Gilly swallowed a goldfish in a booth`
  - ?e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- `Gilly swallowed a goldfish in every booth`
  - ?e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, location(e)), ...

  ?g goldfish(g), swallowee(e,g) ? b booth(b)? location(e,b)

- Does this mean what we'd expect??

  says that there's only one event
  with a single goldfish getting swallowed
  that took place in a lot of booths ...

Quantifier Order

- Groucho Marx celebrates quantifier order ambiguity:
  - **In this country a woman gives birth every 15 min. Our job is to find that woman and stop her.**
  - ∃woman (∀ 15min gives-birth-during(woman, 15min))
  - ∀ 15min (∃woman gives-birth-during(15min, woman))
  - Surprisingly, both are possible in natural language!
  - Which is the joke meaning (where it's always the same woman) and why?

Quantifier Order

- `Gilly swallowed a goldfish in a booth`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- `Gilly swallowed a goldfish in every booth`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, location(e)), ...
    - ∃g goldfish(g), swallowee(e,g)
    - ∀ b booth(b)⇒ location(e,b)
- Does this mean what we'd expect??
  - It's ∃e ∀ b which means same event for every booth
  - Probably false unless Gilly can be in every booth during her swallowing of a single goldfish

Quantifier Order

- `Gilly swallowed a goldfish in a booth`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), exists(booth, location(e)), ...
- `Gilly swallowed a goldfish in every booth`
  - ∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), all(booth, ∃b location(e,b))
- Other reading (∀ b ∃e) involves quantifier raising:
  - all(booth, ∃b [∃e past(e), act(e,swallowing), swallower(e,Gilly), exists(goldfish, swallowee(e)), location(e,b)])
  - "for all booths b, there was such an event in b"

Intensional Arguments

- `Willy wants a unicorn`
  - ∃e act(e,wanting), wanter(e,Willy), exists(unicorn, ∃u wantee(e,u))
    - "there is a unicorn u that Willy wants"
    - here the wantee is an individual entity
  - ∃e act(e,wanting), wanter(e,Willy), wantee(e, ∃u unicorn(u))
    - "Willy wants any entity u that satisfies the unicorn predicate"
    - here the wantee is a type of entity
- `Willy wants Lilly to get married`
  - ∃e present(e), act(e,wanting), wanter(e,Willy), wantee(e, ∃e' [act(e',marriage), marrier(e',Lilly)])
  - "Willy wants any event e in which Lilly gets married"
  - Here the wantee is a type of event
  - Sentence doesn't claim that such an event exists
- Intensional verbs besides `want`: `hope`, `doubt`, `believe`, …

Intensional Arguments

- `Willy wants a unicorn`
  - ∃e act(e,wanting), wanter(e,Willy), wantee(e, ∃g unicorn(g))
    - "Willy wants anything that satisfies the unicorn predicate"
    - here the wantee is a type of entity
- Problem (a fine point I'll gloss over):
  - ∃g unicorn(g) is defined by the actual set of unicorns ("extension")
  - But this set is empty: ∃g unicorn(g) = ∃g FALSE = ∃g dodo(g)
  - Then `wants a unicorn` = `wants a dodo`. Oops!
  - So really the wantee should be criteria for unicornness ("intension")
- Traditional solution involves "possible-world semantics"
  - Can imagine other worlds where set of unicorn ∃ set of dodos
  - Other worlds also useful for:

```
You must pay the rent
You can pay the rent
If you hadn't, you'd be homeless
```


Control

- `Willy wants Lilly to get married`
  - ∃e present(e), act(e,wanting), wanter(e,Willy), wantee(e, ∃f [act(f,marriage), marrier(f,Lilly)])
- `Willy wants to get married`
  - Same as `Willy wants Willy to get married`
  - Just as easy to represent as `Willy wants Lilly` …
  - The only trick is to construct the representation from the syntax. The empty subject position of "to get married" is said to be controlled by the subject of "wants."

Nouns and Their Modifiers

- expert
  - ?g expert(g)
- big fat expert
  - ?g big(g), fat(g), expert(g)
  - But: bogus expert
    - Wrong: ?g bogus(g), expert(g)
    - Right: ?g (bogus(expert))(g) ... bogus maps to new concept
- Baltimore expert (white-collar expert, TV expert …)
  - ?g Related(Baltimore, g), expert(g) – expert from Baltimore
  - Or with different intonation:
    - ?g (Modified-by(Baltimore, expert))(g) – expert on Baltimore
  - Can't use Related for that case: law expert and dog catcher = ?g Related(law,g), expert(g), Related(dog, g), catcher(g) = dog expert and law catcher

Nouns and Their Modifiers

- the goldfish that Gilly swallowed
- every goldfish that Gilly swallowed
- three goldfish that Gilly swallowed

?g [goldfish(g), swallowed(Gilly , g)]

- three swallowed-by-Gilly goldfish (like an adjective!)

Or for real: ?g [goldfish(g), ?e [past(e), act(e,swallowing), swallower(e,Gilly), swallowee(e,g)]]

Adverbs

- Lili passionately wants Billy
  - Wrong?: passionately(want(Lili,Billy)) = passionately(true)
  - Better: (passionately(want))(Lili,Billy)
  - Best: ?e present(e), act(e,wanting), wanter(e,Lili), wantee(e, Billy), manner(e, passionate)
- Lili often stalks Billy
  - (often(stalk))(Lili,Billy)
  - many(day, ?d ?e present(e), act(e,stalking), stalker(e,Lili), stalkee(e, Billy), during(e,d))
- Lili obviously likes Billy
  - (obviously(like))(Lili,Billy) – one reading
  - obvious(likes(Lili , Billy)) – another reading

Speech Acts

- What is the meaning of a full sentence?
  - Depends on the punctuation mark at the end.
  - Billy likes Lili. → **assert**(like(B,L))
  - Billy likes Lili? → **ask**(like(B,L))
    - or more formally, "Does Billy like Lili?"
  - Billy, like Lili! → **command**(like(B,L))
- Let's try to do this a little more precisely, using event variables etc.

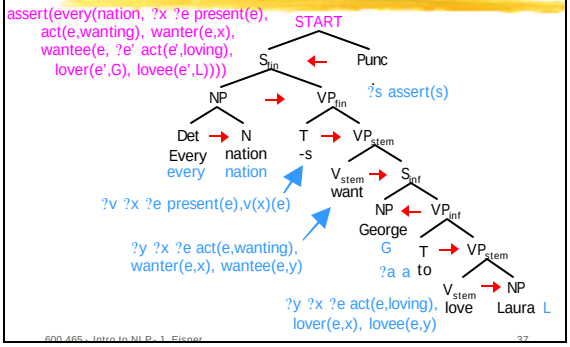
Speech Acts

- What did Gilly swallow?
  - **ask**(?x ?e past(e), act(e,swallowing), swallower(e,Gilly), swallowee(e,x))
  - Argument is identical to the modifier "that Gilly swallowed"
  - Is there any common syntax?
- Eat your fish!
  - **command**(?f act(f,eating), eater(f,Hearer), eatee(...))
- I ate my fish.
  - **assert**(?e past(e), act(e,eating), eater(f,Speaker), eatee(...))

Compositional Semantics

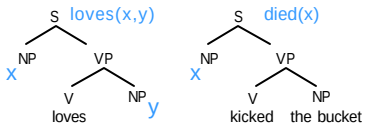
- We've discussed what semantic representations should look like.
- **But how do we get them from sentences???**
- First - parse to get a syntax tree.
- Second - look up the semantics for each word.
- Third - build the semantics for each constituent
  - Work from the bottom up
  - The syntax tree is a "recipe" for how to do it

Compositional Semantics



Compositional Semantics

- Add a "sem" feature to each context-free rule
  - S ? NP loves NP
  - S[sem=loves(x,y)] ? NP[sem=x] loves NP[sem=y]
  - Meaning of S depends on meaning of NPs
- TAG version:

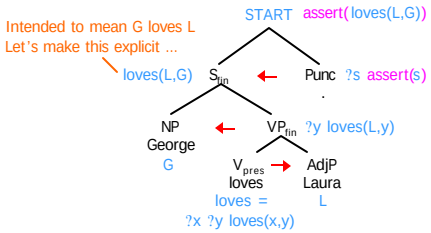


- Template filling: S[sem=showflights(x,y)] ? I want a flight from NP[sem=x] to NP[sem=y]

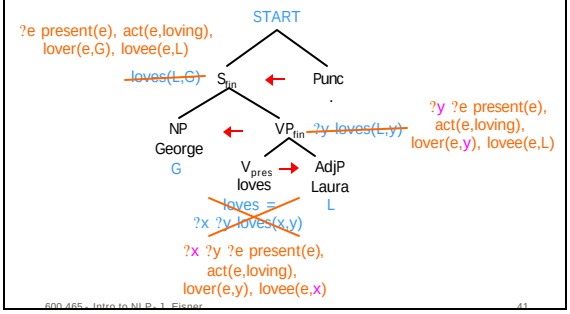
Compositional Semantics

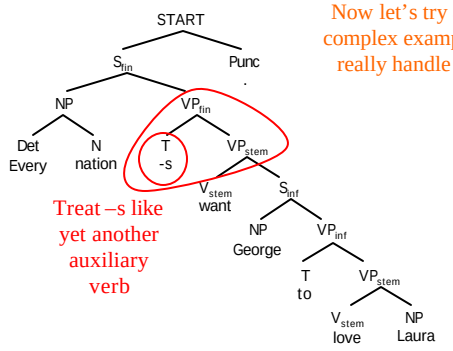
- Instead of S ? NP loves NP
  - S[sem=loves(x,y)] ? NP[sem=x] loves NP[sem=y]
- might want general rules like S ? NP VP:
  - V[sem=loves] ? loves
  - VP[sem=v(obj)] ? V[sem=v] NP[sem=obj]
  - S[sem=vp(subj)] ? NP[sem=subj] VP[sem=vp]
- Now George loves Laura has sem=loves(Laura)(George)
- In this manner we'll sketch a version where
  - Still compute semantics bottom-up
  - Grammar is in Chomsky Normal Form
  - So each node has 2 children: 1 function & 1 argument
  - To get its semantics, apply function to argument!
  - *(version on homework will be a little less pure)*

Compositional Semantics

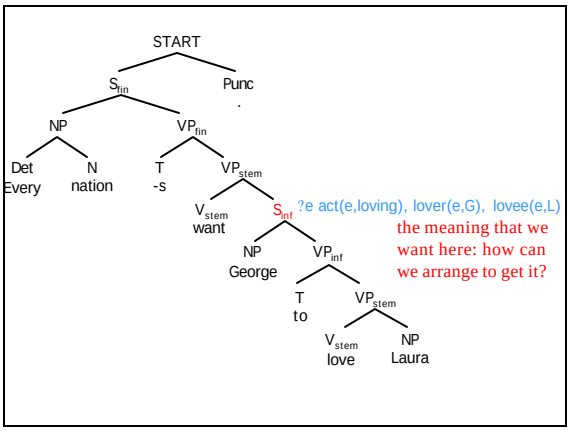


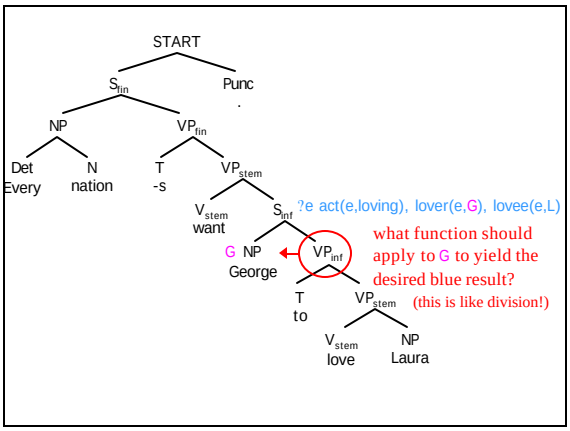
Compositional Semantics

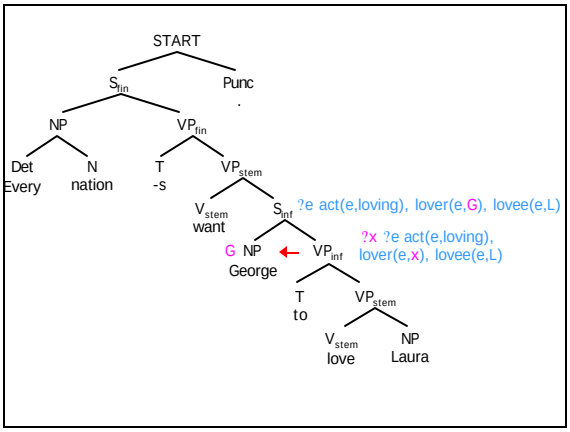


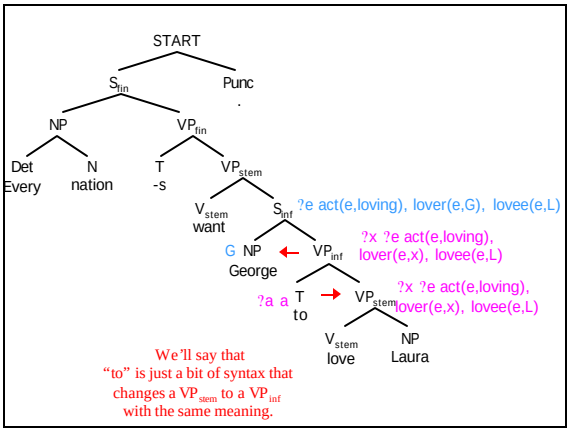


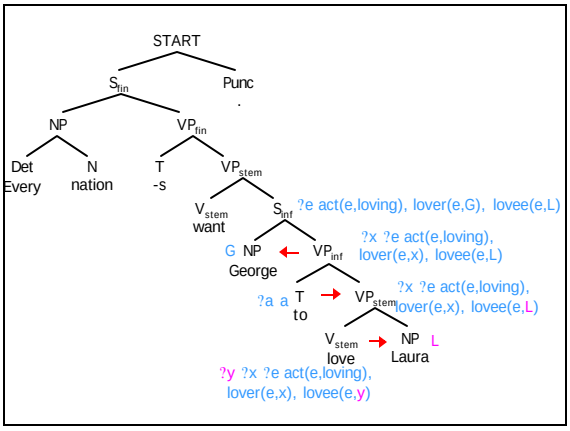
Now let's try a more complex example, and really handle tense.

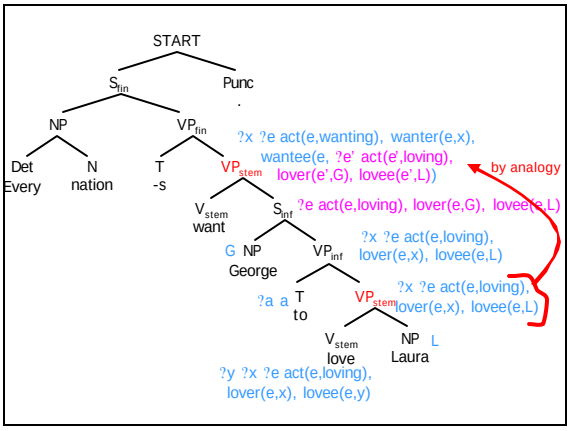

START
Punc
.
Det
Every
N
nation
T
-s
want
George
to
love
Laura
?e act(e,loving), lover(e,G), lovee(e,L)
the meaning that we want here: how can we arrange to get it?

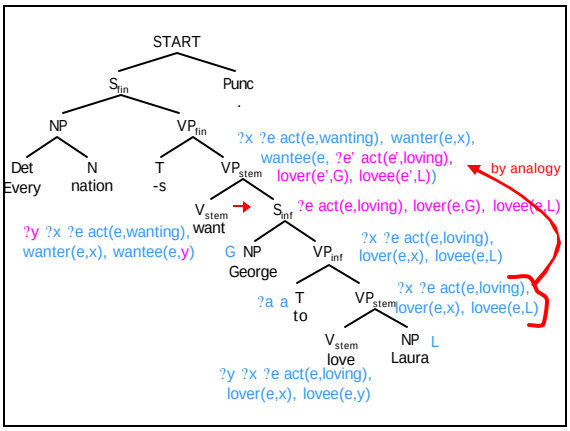

START
Punc
.
Det
Every
N
nation
T
-s
want
G
George
to
love
Laura
?e act(e,loving), lover(e,G), lovee(e,L)
what function should apply to G to yield the desired blue result?
(this is like division!)

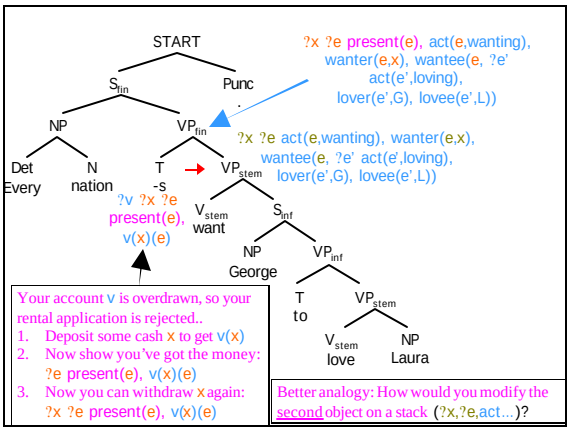

START
Punc
.
Det
Every
N
nation
T
-s
want
G
George
to
love
Laura
?e act(e,loving), lover(e,G), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)

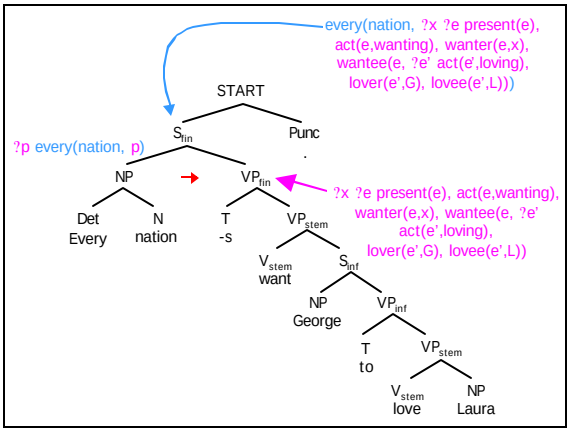

START
Punc
.
Det
Every
N
nation
T
-s
want
G
George
?a a
to
love
Laura
?e act(e,loving), lover(e,G), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
We'll say that "to" is just a bit of syntax that changes a VP stem to a VP inf with the same meaning.

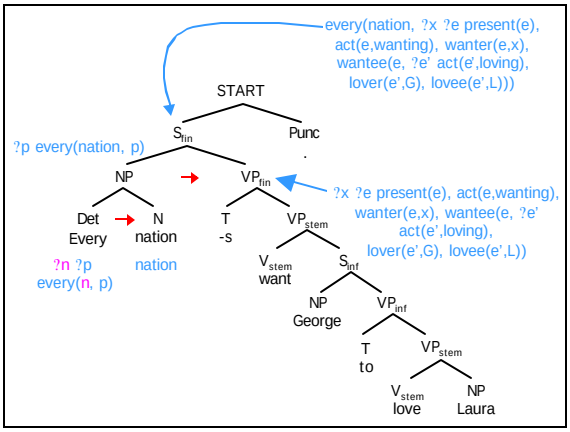

START
Punc
.
Det
Every
N
nation
T
-s
want
G
George
?a a
to
love
Laura
L
?e act(e,loving), lover(e,G), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?y ?x ?e act(e,loving), lover(e,x), lovee(e,y)

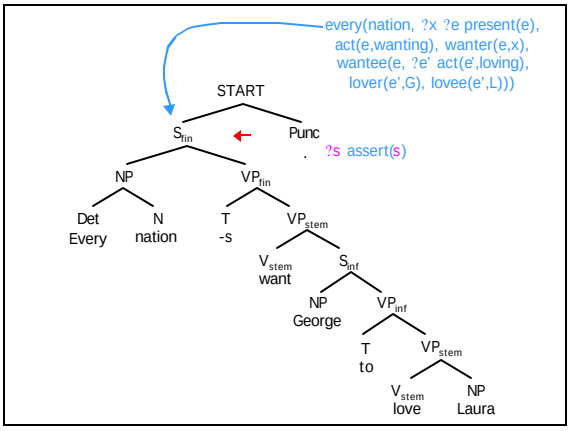

START
Punc
.
Det
Every
N
nation
T
-s
want
G
George
?a a
to
love
Laura
L
?x ?e act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving),
lover(e',G), lovee(e',L))
by analogy
?e act(e,loving), lover(e,G), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?y ?x ?e act(e,loving), lover(e,x), lovee(e,y)

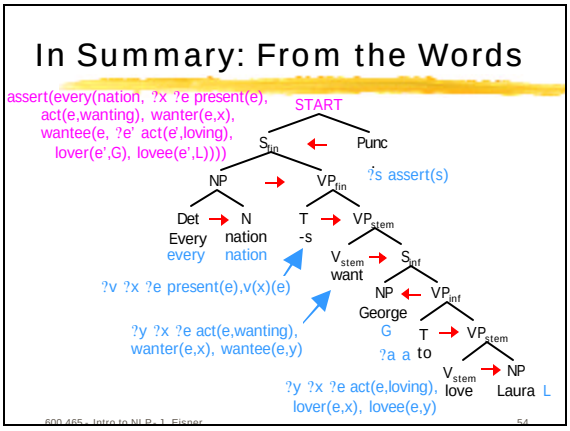

START
Punc
NP
Det
Every
N
nation
T
-s
by analogy
?x ?e act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))
want
?e act(e,loving), lover(e,G), lovee(e,L)
?y ?x ?e act(e,wanting), wanter(e,x), wantee(e,y)
G
George
?x ?e act(e,loving), lover(e,x), lovee(e,L)
?a a
to
love
Laura
L
?y ?x ?e act(e,loving), lover(e,x), lovee(e,y)


START
Punc
NP
Det
Every
N
nation
T
-s
?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))
?x ?e act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))
?v ?x ?e present(e), v(x)(e)
want
George
to
love
Laura
Your account v is overdrawn, so your rental application is rejected..
1. Deposit some cash x to get v(x)
2. Now show you've got the money: ?e present(e), v(x)(e)
3. Now you can withdraw x again: ?x ?e present(e), v(x)(e)
Better analogy: How would you modify the second object on a stack (?x,?e,act...)?


every(nation, ?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L)))
START
?p every(nation, p)
NP
Det
Every
N
nation
T
-s
?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))
want
George
to
love
Laura


every(nation, ?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L)))
START
?p every(nation, p)
NP
Det
Every
N
nation
?n ?p every(n, p)
nation
T
-s
?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))
want
George
to
love
Laura


every(nation, ?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L)))
START
Punc
?s assert(s)
NP
Det
Every
N
nation
T
-s
want
George
to
love
Laura


In Summary: From the Words
assert(every(nation, ?x ?e present(e), act(e,wanting), wanter(e,x), wantee(e, ?e' act(e',loving), lover(e',G), lovee(e',L))))
START
Punc
?s assert(s)
NP
Det
Every
every
N
nation
nation
T
-s
want
George
G
?a a
to
love
Laura
L
?v ?x ?e present(e),v(x)(e)
?y ?x ?e act(e,wanting), wanter(e,x), wantee(e,y)
?y ?x ?e act(e,loving), lover(e,x), lovee(e,y)
600.465 - Intro to NLP - J. Eisner
54

Other Fun Semantic Stuff: A Few Much-Studied Miscellany

- Temporal logic
  - Gilly had swallowed eight goldfish before Milly reached the bowl
  - Billy said Jilly was pregnant
  - Billy said, "Jilly is pregnant."
- Generics
  - Typhoons arise in the Pacific
  - Children must be carried
- Presuppositions
  - The king of France is bald.
  - Have you stopped beating your wife?
- 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.