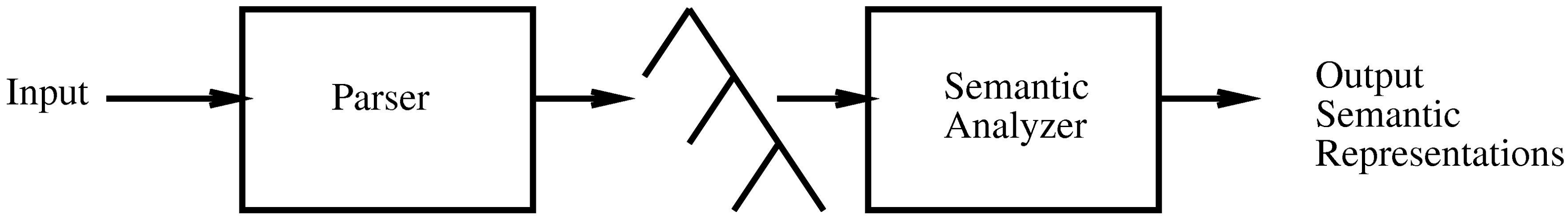
CS4025: Logic-Based Semantics

- Compositionality in practice
- Producing logic-based meaning representations
- Use of the lambda calculus
- Problems with quantification

See J&M chap 15 in 1st ed, 18 in 2nd

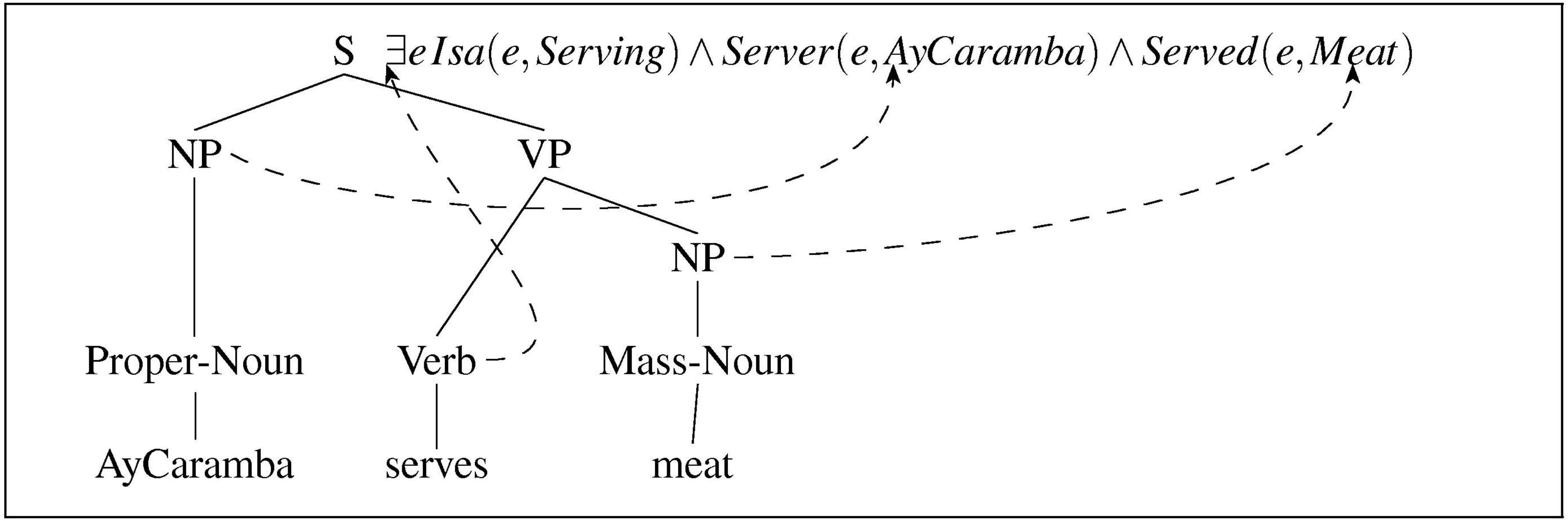
Two stage approach to NLU



Compositionality

- "The meaning of the whole is a function of the meanings of the parts"
  - A necessary assumption if we want a *general* approach to meaning extraction beyond a listing of sentences with their meanings
  - What are the parts? It's up to our grammar to yield appropriate parts
  - What are the functions and what types of meanings do the parts have? These are very challenging questions…

Result of semantic analysis



How to get there compositionally?

Expressing semantic rules

- *Associate semantic information with each grammar rule – semantic attachments.*
- These say how the semantics of the whole phrase is computed from the semantics of the parts

A → a1 a2 … a2 { f(a1.sem,a2.sem,a3.sem) }

This says that the semantics of an *A* recognised by this rule is function f of the semantics of the parts a1 a2 … an (ai.sem = semantics of ai)

Example – English number grammar

Num → Digit hundred and Ty Digit

{ 100*Digit1.sem + Ty.sem + Digit2.sem }

Digit → two { 2 }

Digit → three { 3 }

…

Ty → twenty { 20 }

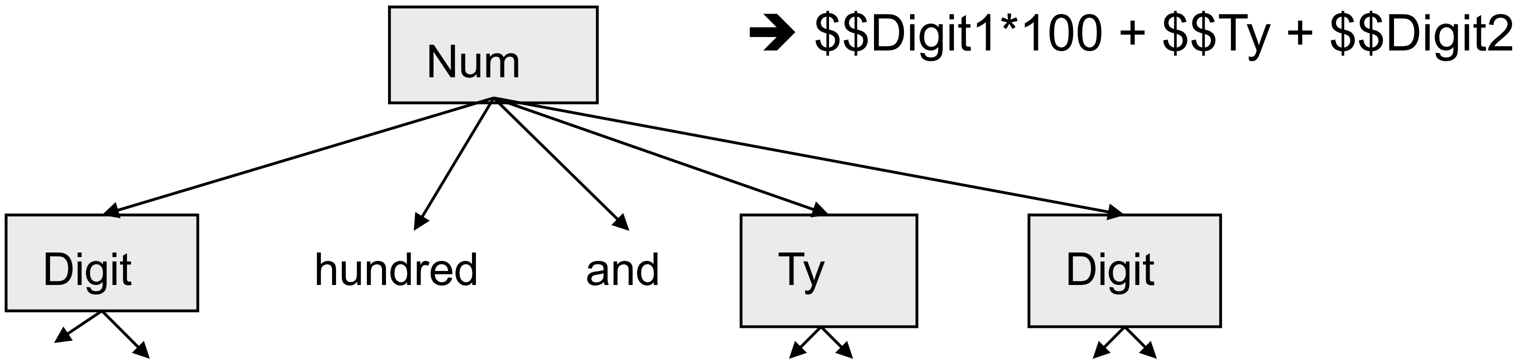
Ty → thirty { 30 }

…

What is the semantics of “two hundred and thirty two”?

Alternative Approach

- Instead of putting semantic attachments with grammar rules, write rules for translating *parse trees*.



- This is equivalent (as each use of a grammar rule corresponds to a node in the parse tree)

Initial semantic attachments for English

ProperNoun → AyCaramba **{ AyCaramba }**

MassNoun → meat **{ Meat }**

Verb → serves

$$\{\exists \mathbf{e,x,y}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,y)}\}$$

NB note caseframe approach of logic representation (makes handling, e.g. adverbs easier)

How do these combine together to give meanings for the larger phrases?

Propagating meanings up NPs

NP → ProperNoun **{ ProperNoun.sem }**

NP → MassNoun **{ MassNoun.sem }**

- When a phrase just has one part, often just return the meaning of this as the meaning of the whole phrase

VP meanings

- What should the meaning of “serves Meat” be?

  Arguably, something like:

  $\exists \mathbf{e,x}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,Meat)}$
- How do we compute this from the Verb and NP meanings:

  Verb:

  $\exists \mathbf{e,x,y}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,y)}$

  NP: **Meat**
- Need to somehow “incoporate” the NP meaning into the Verb meaning

Basic Idea

- Represent the Verb meaning as having a "hole" where the NP meaning will go:

$$\exists \mathbf{e,x}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,}\otimes\mathbf{)}$$

- Create the VP meaning by filling the hole:

$$\exists \mathbf{e,x}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,Meat)}$$

- Informal grammar rule:

  VP → V NP { **fillHole(V.sem,NP.sem)** }

The Lambda Calculus

- The Lambda Calculus is an extension to Predicate Calculus that allows us to have "named holes"
- The above Verb meaning would be represented as:

$$\lambda \mathbf{y}\ \exists \mathbf{e,x}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,y)}$$

(or could have any other name instead of **y**)

- Filling the hole in X with Y is called *applying* X to Y and is denoted X(Y)
- VP → Verb NP **{ Verb.sem(NP.sem) }**

Examples of Application

$$(\lambda x\, P(x, y))(apple) = P(apple, y)$$

$$(\lambda x\, P(y, x))(apple) = P(y, apple)$$

$$(\lambda z\, P(z, x))(apple) = P(apple, x)$$

Computing the S meaning

We now have

NP meaning: **AyCaramba**

VP meaning:

$$\exists \mathbf{e,x}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,x)} \wedge \mathbf{Served(e,Meat)}$$

How to combine to get the S meaning:

$$\exists \mathbf{e}\ \mathbf{Isa(e,Serving)} \wedge \mathbf{Server(e,AyCaramba)} \wedge \mathbf{Served(e,Meat)}$$

This is the same problem as before! Now the Verb meaning needs to have *two* holes…

Basic Idea

- Order the holes in a formula according to the order in which they are going to be filled:

$$\exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e,} \otimes_2\mathbf{)} \wedge \mathbf{Served(e,} \otimes_1\mathbf{)}$$

- First the hole corresponding to the object of the verb will be filled, then the subject

In Lambda Calculus

- Use nesting (and different variable names) to indicate order of holes – the outermost name corresponds to the first hole to be filled

$$\lambda x\ \lambda y\ \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, y)} \wedge \mathbf{Served(e, x)}$$

$$(\lambda x\ \lambda y\ \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, y)} \wedge \mathbf{Served(e, x)}\ )\mathbf{(Meat)}$$
$$= \lambda y\ \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, y)} \wedge \mathbf{Served(e, Meat)}$$

$$(\lambda y\ \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, y)} \wedge \mathbf{Served(e, Meat)})\mathbf{(AyC)}$$
$$= \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, AyCaramba)} \wedge \mathbf{Served(e, Meat)}$$

Examples with nested and complex lambda expressions

$$((\lambda x\ \lambda y\ P(x,y))(apple))(pear) = P(apple,pear)$$

$$((\lambda x\ \lambda y\ P(y,x))(apple))(pear) = P(pear,apple)$$

$$(\lambda x\ P(x(apple),y))(\lambda z\ f(z,z)) = P(f(apple,apple),y)$$

Revised Verb, VP, S rules

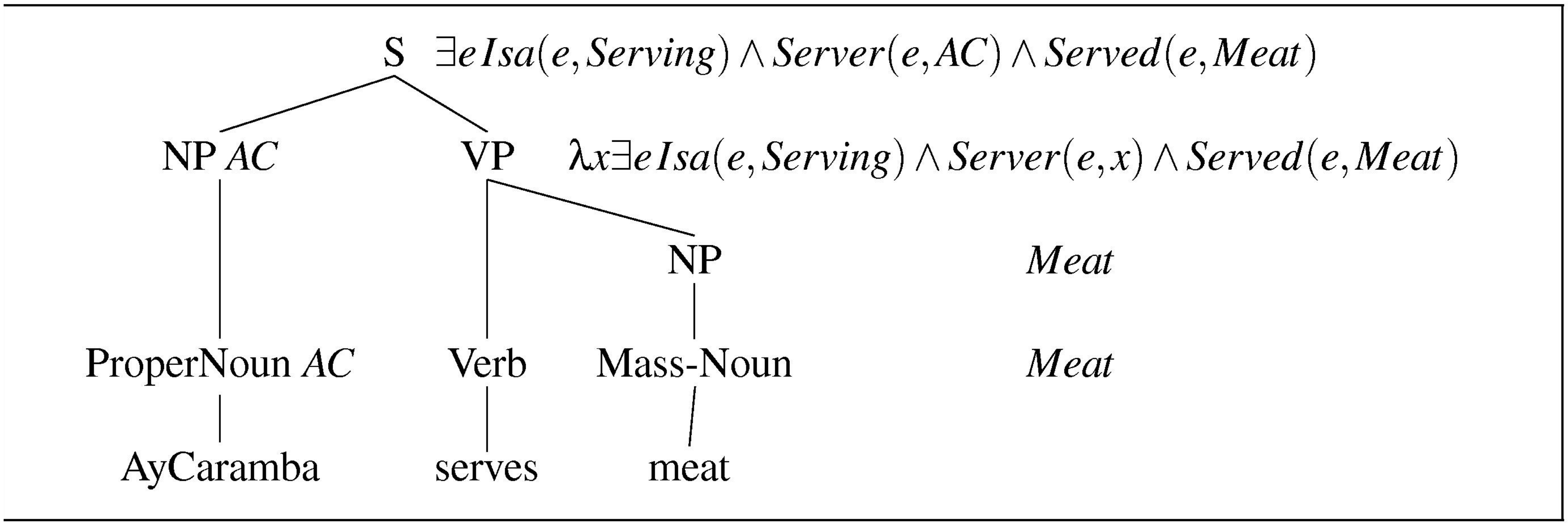
Verb → serves

$\{\lambda x\ \lambda y\ \exists e\ Isa(e, Serving) \wedge Server(e, y) \wedge Served(e, x)\}$

VP → Verb NP { **Verb.sem(NP.sem)** }

S → NP VP { **VP.sem(NP.sem)** }

Final result



More complex NPs

- Some possible rules:

NP → every N $\{\forall x\ Isa(x, N.sem)\}$

NP → a N $\{\exists x\ Isa(x, N.sem)\}$

N → restaurant $\{Restaurant\}$

- But what about “every restaurant serves meat”? We get:

$$\exists e\ Isa(e, Serving) \wedge Server(e, \forall x\ Isa(x, Restaurant)) \wedge Served(e, Meat)$$

Dealing with Quantification (1a)

- Since the quantifiers coming from NPs need to be at the outside of the S meaning, put the NP in charge, rather than the VP:

S → NP VP **{ NP.sem(VP.sem) }**

NP → every N **{ λv $\forall$z Isa(z,N.sem) $\supset$ v(z) }**

VP etc as before

Dealing with Quantification (1b)

NP semantics: $\{ \lambda v\ \forall z\ \mathbf{Isa(z, Restaurant)} \supset \mathbf{v(z)} \}$

VP semantics (as before):

$\{ \lambda x\ \exists e\ \mathbf{Isa(e, Serving)} \wedge \mathbf{Server(e, x)} \wedge \mathbf{Served(e, Meat)} \}$

S semantics:

$$(\lambda v\ \forall z\ \mathrm{Isa}(z, \mathrm{Restaurant}) \supset v(z))($$
$$\lambda x\ \exists e\ \mathrm{Isa}(e, \mathrm{Serving}) \wedge \mathrm{Server}(e, x) \wedge \mathrm{Served}(e, \mathrm{Meat}))$$
$$= \forall z\ \mathrm{Isa}(z, \mathrm{Restaurant}) \supset$$
$$(\lambda x\ \exists e\ \mathrm{Isa}(e, \mathrm{Serving}) \wedge \mathrm{Server}(e, x) \wedge \mathrm{Served}(e, \mathrm{Meat})))(z))$$
$$= \forall z\ \mathrm{Isa}(z, \mathrm{Restaurant}) \supset$$
$$(\exists e\ \mathrm{Isa}(e, \mathrm{Serving}) \wedge \mathrm{Server}(e, z) \wedge \mathrm{Served}(e, \mathrm{Meat}))$$

Dealing with Quantification (2)

- Leave the placing of quantifiers for a post-processing stage.
- This means allowing quantifiers where terms would normally be expected – called *complex-terms*

NP → every N $\quad \{ < \forall z \; \mathbf{Isa(z, N.sem)} > \}$

S meaning from original rules:

$$\exists e \; \mathbf{Isa(e, Serving)} \wedge$$
$$\mathbf{Server(e,} < \forall x \; \mathbf{Isa(x, Restaurant)} > \mathbf{)} \wedge \mathbf{Served(e, Meat)}$$

Handling Quantifier Ambiguity

- By the second method, "every restaurant serves a salad" comes out as:

$$\exists e\ \mathbf{Isa(e, Serving)} \wedge$$
$$\mathbf{Server(e,} < \forall\ \mathbf{z\ Isa(z, Restaurant)} > \mathbf{)} \wedge$$
$$\mathbf{Served(e,} < \exists\ \mathbf{w\ Isa(w, Salad)} > \mathbf{)}$$

- The two quantifiers can be moved to the outside of the formula in different orders, corresponding to different interpretations
- The *Hobbs-Shieber* algorithm enumerates all possibilities

Formal Semantics

- The field of "formal semantics" studies how to model many phenomena of natural language in this kind of way
- One inspiration was the work of the logician Montague in the 1970's ("Montague grammar"). Montagu was the first person to suggest this kind of formal approach to NL semantics
- We have presented just the tip of the iceberg of a large research area