Meaning as computation?

Computational semantics seminar

January 18, 2017

This course

Computation in semantics

This course is about **computation in semantics**.

Not in the sense of parsing, or machine learning, or anything in that neighborhood really.

Instead: how thinking about **meaning as computation** enriches our theory of meaning.

Why meaning as computation?

I think taking this perspective is likely to be fruitful:

- What programmers do when they reason about the semantics of programming languages is *remarkably* similar to what linguists do when we reason about the semantics of natural languages.

As we will see in some detail today, some of the problems linguists care about have direct, well-studied analogs in programming contexts!

Abstract thinking

Programmers working in richly typed languages often remark that their programs tend to "just work" once they pass the typechecker, much more often than they feel they have a right to expect. One possible explanation for this is that not only trivial mental slips (e.g., forgetting to convert a string to a number before taking its square root), but also deeper conceptual errors (e.g., neglecting a boundary condition in a complex case analysis, or confusing units in a scientific calculation), will often manifest as inconsistencies at the level of types.

(Pierce 2002: 4)

Some things we will explore

- Interpreters and extending interpreters: monads
- Interpretation as compilation
- Continuations
- Type-theoretic aspects of meaning and composition
- Fragments and implementing analyses

The work for this course will involve

Regular reading (see the course website)

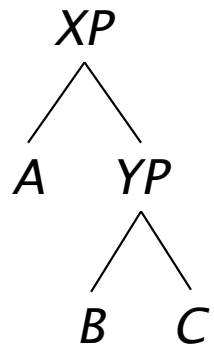
Periodic written work (practicing the formal tools)

And a short term paper at the end (10–15pp)

Some semantics

A baseline semantic theory

Let's imagine we have a tree with some terminals *A*, *B*, *C*, and some binary-branching nodes *XP*, *YP*, as follows:



Our task as semanticists: deriving a meaning for *XP*, in terms of the meanings of its parts, and the way those parts are put together.

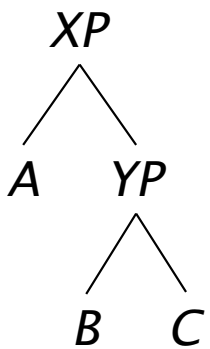
- This is the folk sense of *compositionality*.

Montague's solution

Define a recursive interpretation function $[\![\cdot]\!]$, as follows:

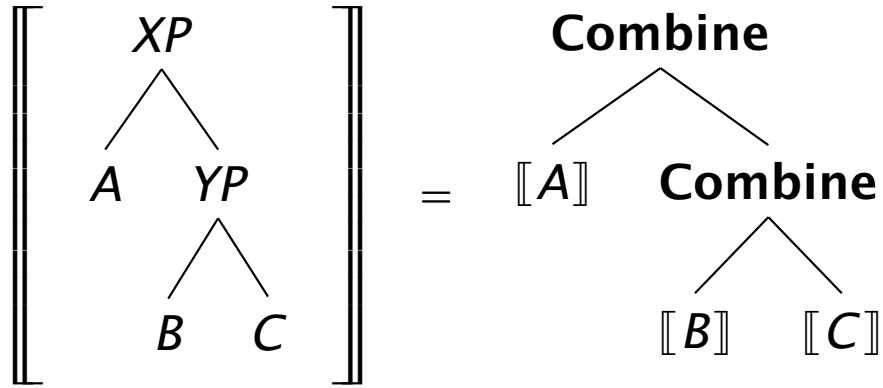
$$[\![[_{XP}\ \Gamma\ \Delta]]\!] := \textbf{Combine}\,([\![\Gamma]\!], [\![\Delta]\!])$$

Then, the interpretation for *XP* is calculated as follows:



$$\begin{aligned}[\![XP]\!] &= \textbf{Combine}\,([\![A]\!], [\![YP]\!]) \\ &= \textbf{Combine}\,([\![A]\!], \textbf{Combine}\,([\![B]\!], [\![C]\!]))\end{aligned}$$

Taking a higher view



Thus, $[\![\cdot]\!]$ is a structure-preserving map (i.e., a homomorphism) between the syntactic and semantic algebras.

- This, of course, was Montague's sense of compositionality.

Some questions

Of course, at this point we have said very, very little!

A couple questions we can ask about this setup:

- What is the nature of $[\![A]\!]$, $[\![B]\!]$, and $[\![C]\!]$?
- What is the nature of **Combine**?

Standard answer, Part 1

First, we specify what kinds of meanings we're interested in (and draw lexical entries — i.e., denotations for terminal nodes)

$$\tau ::= e \mid t \mid \tau \rightarrow \tau$$

This means that the set of NL types τ consists of e (individuals), t (truth-values), and (the inductive step) functions mapping things drawn from τ into other things drawn from τ.

This way of notating types is known as BNF (for Backus-Naur Form).

The standard answer, Part 2

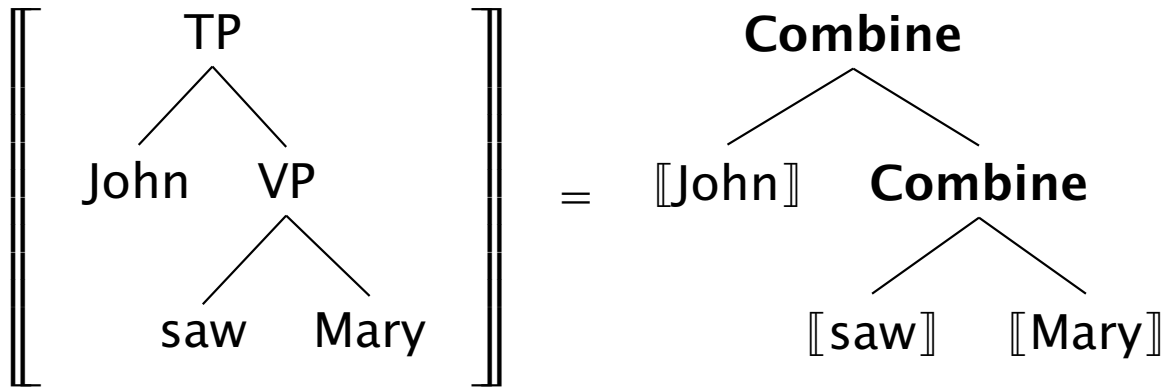
Second, we provide a more detailed characterization of **Combine**:

$$\textbf{Combine}\,(A, B) := \begin{cases} A\,B & \textit{if } A :: \sigma \to \tau, B :: \sigma \quad \text{(FA)} \\ B\,A & \textit{if } A :: \sigma, B :: \sigma \to \tau \quad \text{(BA)} \\ \lambda x.\, A\,x \wedge B\,x & \textit{if } A, B :: \sigma \to t \quad \text{(PM)} \end{cases}$$

Thus, **Combine** does one of three operations — forward application, backward application, or predicate modification — depending on the types of the expressions to be combined.

- This is known as *type-driven* composition (Klein & Sag 1985).

An example



Which ends up reducing to ⟦saw⟧⟦Mary⟧⟦John⟧.

- ⟦John⟧ and ⟦Mary⟧ are the individuals John and Mary, type e.
- ⟦saw⟧ is the $f : e \to e \to t$ mapping two individuals to $\mathbb{T}$ iff they stand in the seeing relation.

Ignorance was bliss

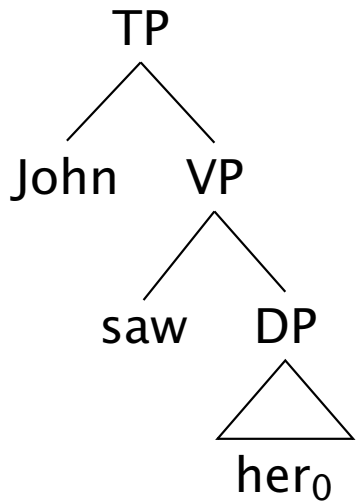
We can do some things with this semantics, but there is so much we *cannot* yet handle:

- Pronouns and pronominal binding
- Presupposition (projection)
- Questions (qua sets of alternative propositions)
- (Association with) focus
- Supplemental content (and projection)
- Quantification (and scope)

We're going to spend some time today going through these various enrichments. With one very notable exception, the solutions seem to bear a certain abstract similarity to each other!

Some enrichments

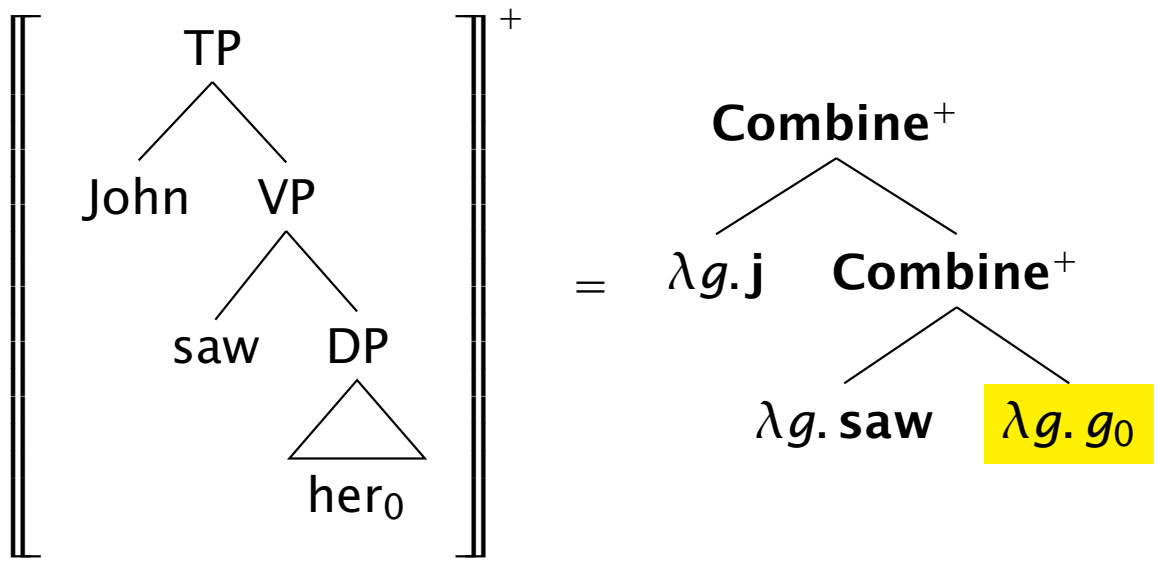
Binding/intensionality



$$\tau^{+} ::= s \to \tau$$

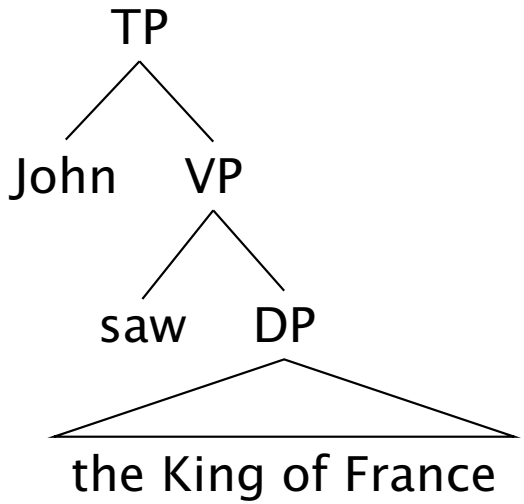
$$[\![\Gamma\ \Delta]\!]^{+} := \lambda g.\, \textbf{Combine}\,([\![\Gamma]\!]\, g, [\![\Delta]\!]\, g)$$

Binding example



$$= \lambda g.\ \textbf{saw}\ g_0\ \textbf{j}$$

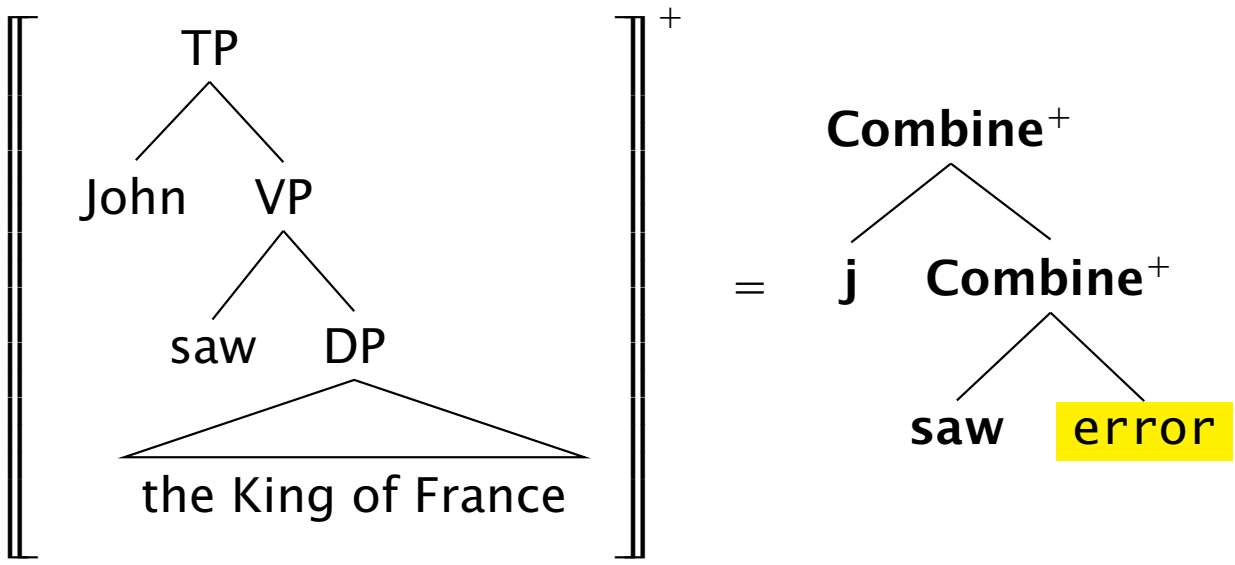
Presuppositions



$$\tau^{+} ::= \tau \mid \mathtt{Error}\,\tau$$

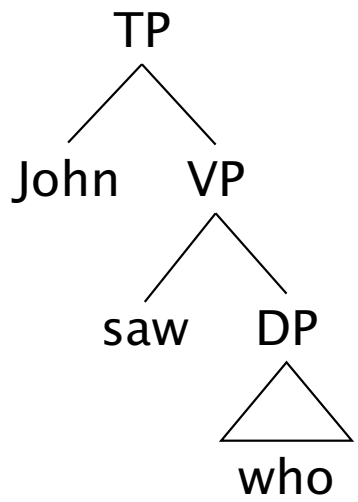
$$[\![\Gamma\ \Delta]\!]^{+} := \begin{cases} \mathtt{error} & \textit{if } [\![\Gamma]\!] = \mathtt{error} \\ \mathtt{error} & \textit{if } [\![\Delta]\!] = \mathtt{error} \\ \textbf{Combine}\,([\![\Gamma]\!], [\![\Delta]\!]) & \textit{otherwise} \end{cases}$$

Example



= error

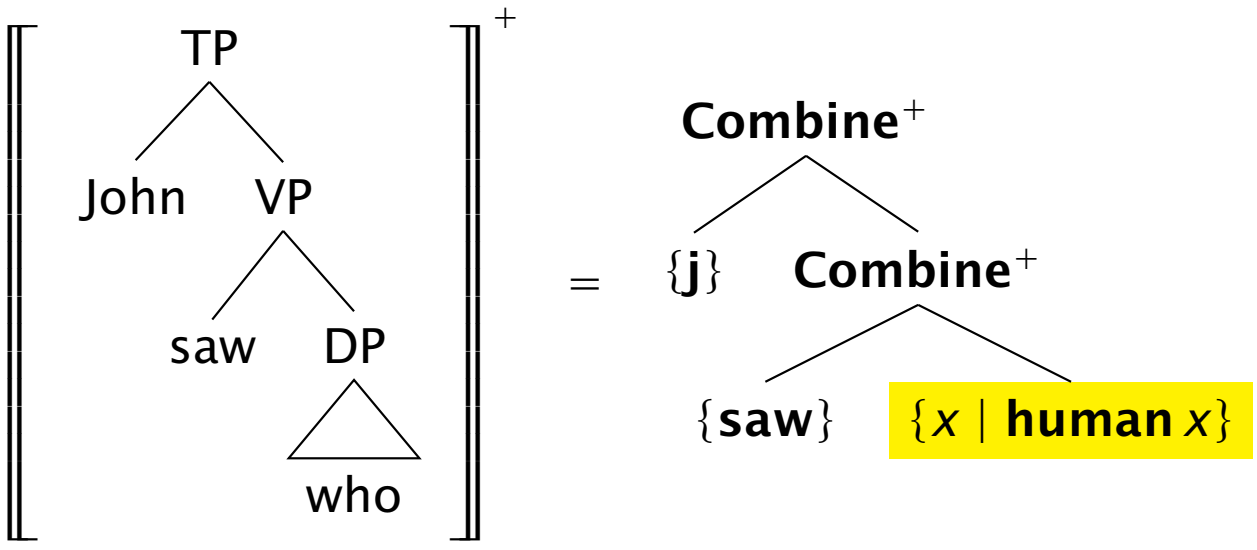
Alternatives



$$\tau^{+} ::= \mathsf{S}\,\tau$$

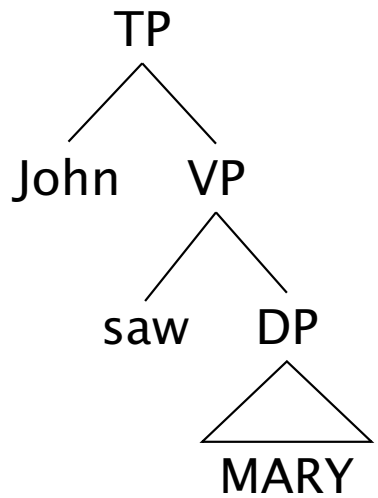
$$[\![\Gamma\ \Delta]\!]^{+} := \{\textbf{Combine}\,(\gamma, \delta) \mid \gamma \in [\![\Gamma]\!],\ \delta \in [\![\Delta]\!]\}$$

Example



$$= \{\textbf{saw}\ x\ \textbf{j} \mid \textbf{human}\ x\}$$

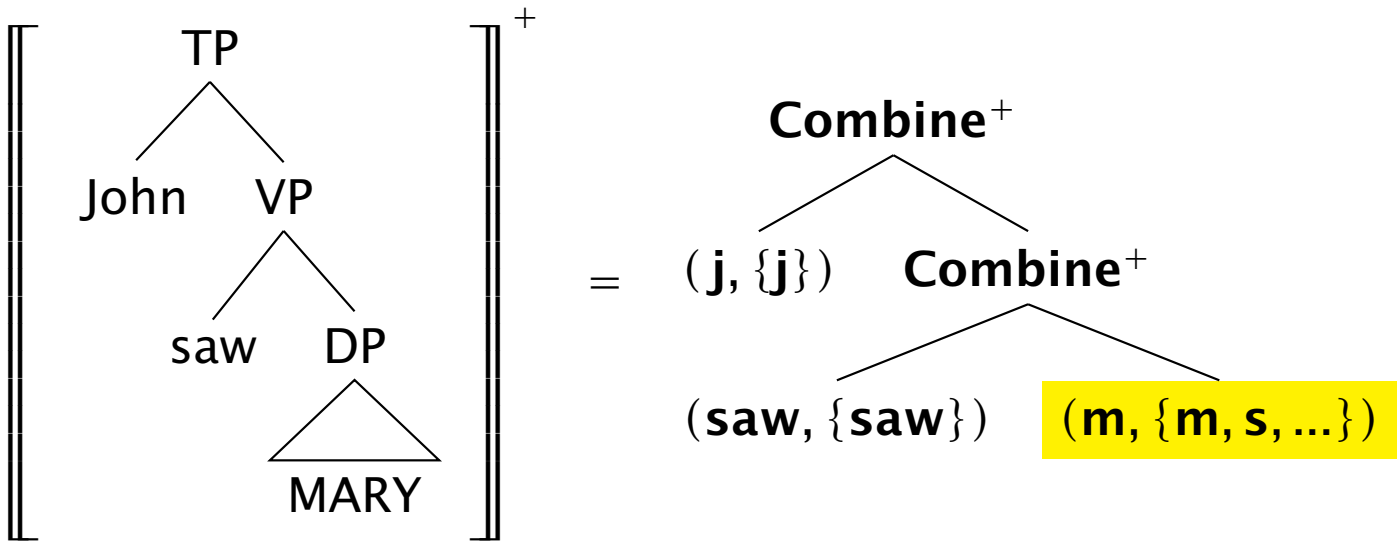
Focus



$$\tau^{+} ::= \tau \times \mathsf{S}\,\tau$$

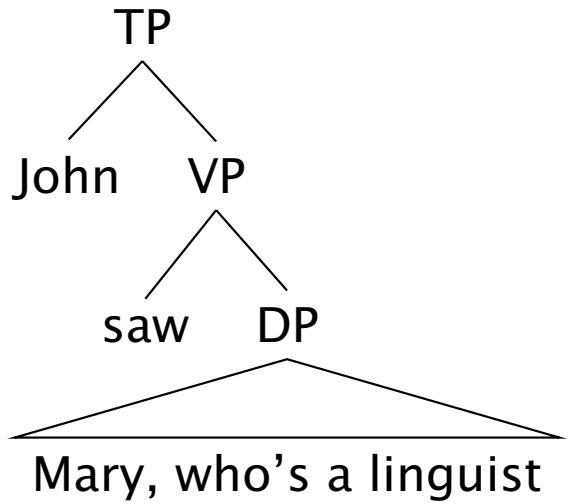
$$[\![\Gamma\ \Delta]\!]^{+} := \Big(\textbf{Combine}\,([\![\Gamma]\!]_o, [\![\Delta]\!]_o),\ \{\textbf{Combine}\,(\gamma, \delta) \mid \gamma \in [\![\Gamma]\!]_f,\ \delta \in [\![\Delta]\!]_f\}\Big)$$

Example



$$= \left(\mathbf{saw\,m\,j},\ \{\mathbf{saw}\ x\ \mathbf{j} \mid x \in \{\mathbf{m}, \mathbf{s}, \ldots\}\}\right)$$

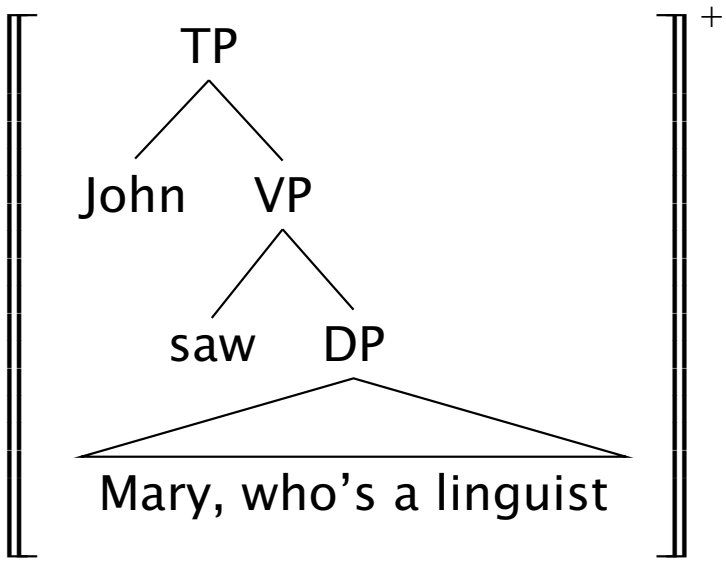
Supplemental content



$$\tau^{+} ::= \tau \times t$$

$$[\![\Gamma\ \Delta]\!]^{+} := \Big(\textbf{Combine}\,([\![\Gamma]\!]_o, [\![\Delta]\!]_o),\ [\![\Gamma]\!]_s \wedge [\![\Delta]\!]_s\Big)$$

Example



$=$ **Combine**$^+$ (**j**, $\mathbb{T}$) **Combine**$^+$ (**saw**, $\mathbb{T}$) (**m**, **ling m**)

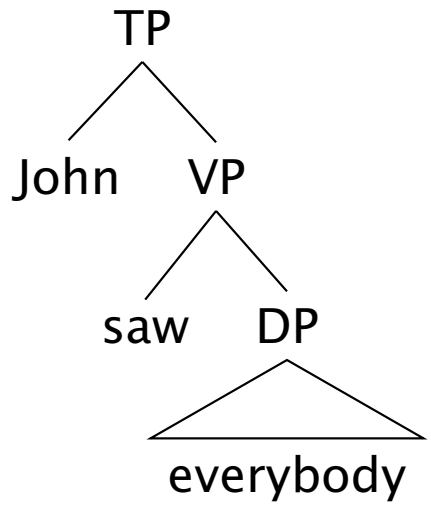
$= ($ **saw m j**, **ling m** $)$

What these have in common

I sense an abstraction lurking here. Do you?

What are some things that you see as commonalities between these different ways to enrich a grammar?

Scope

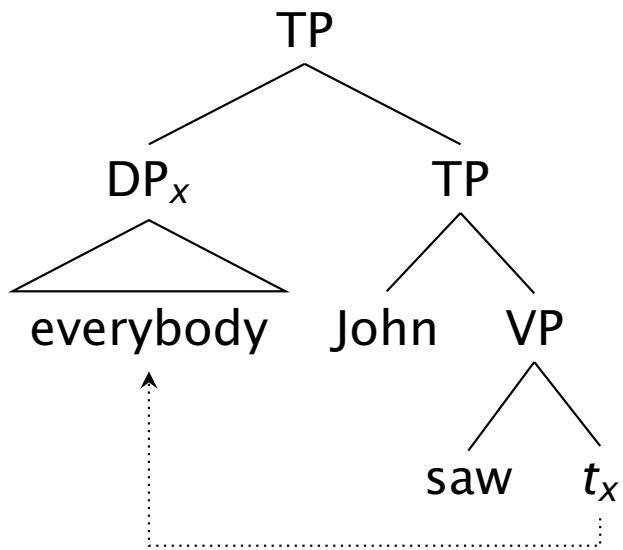


The problem of scope-taking: some expressions need access to more than their immediate semantic context:

$$\textbf{everybody}\ (\lambda x.\ \textbf{saw}\ x\ \textbf{j})$$

Hm!

Is this generally handled in the same way as the other enrichments we've seen so far? Nope! Gets a sui generis treatment:



(Now, we do need a way to establish a link between the raised quantifier and its trace. Generally, this is accomplished by exploiting the Binding apparatus developed earlier.)

Interactions and scaling up

Each of these enrichments is a wonderful, insightful way to think about a given problem domain.

But it's less clear it scales up as well as you'd like.

- Are these enrichments really well suited to their target empirical domains?
- How can we *combine* enrichments? Can we do so?

Case study: Alternatives in semantics

Some data: indefinites

Indefinites have predicative uses:

- Mary is **a linguist**.

Along with quantificational uses:

- Many people know **a linguist**.

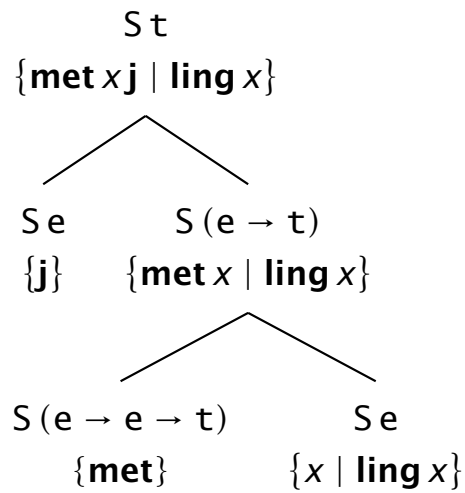
A standard approach to this data has been to take the predicative use of indefinites as basic, and define an operation mapping the predicative meaning into a quantificational meaning (Partee 1986):

$$[\![\text{a linguist}]\!] = \{x \mid \textbf{ling}\, x\}$$
$$\textbf{A} := \lambda S.\, \lambda f.\, \exists x \in S : f\, x$$

Thus, $\textbf{A}\,[\![\text{a linguist}]\!] = \lambda f.\, \exists f \in \{x \mid \textbf{ling}\, x\} : f\, x$

Alternative semantics

Another approach is to take the predicative use as “basic”, and then to upgrade composition into an alternative-friendly mechanism:



This should look familiar from our analysis of questions. Kratzer & Shimoyama (2002) propose something like this for *indeterminate pronouns* in Japanese.

Some properties: islands

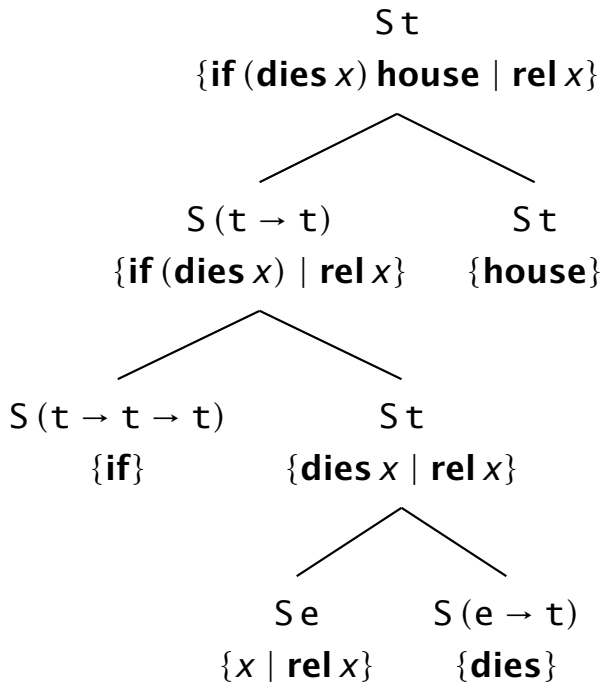
Indefinites have a characteristic ability to project their scope out of domains that other operators cannot (Fodor & Sag 1982, Reinhart 1997):

- If [a rich relative of mine dies], I'll inherit a house. $\exists \gg$ *if*
- Each student has to come up with three arguments showing that [some condition proposed by Chomsky is wrong]. $\forall \gg \exists \gg 3$

True quantifiers (universals, negative existentials, etc.) in parallel positions are trapped inside the bracketed constituent. These domains are known as **scope islands**.

Island-escaping in alternative semantics

Guess what: this behavior is predicted by alternative semantics!



[This was the original motivation for alternative semantics (Rooth 1985). (Hamblin 1973 invented alternative semantics for questions, but there were at the time no other ways to derive sets of propositions.)]

Welp

Actually, nobody has really advocated taking an alternative-semantic perspective on (English) indefinites and their exceptional scope properties. Why not?

Turns out that there seem to be insuperable obstacles to doing so.

Question: selectivity

Two indefinites on an island have the ability to take exceptional scope in different ways outside the island.

- If [a renowned expert on tax law visits a rich relative of mine], I'll inherit a house. $^{\checkmark}\exists_{lawyer} \gg \mathit{if} \gg \exists_{relative}$, $^{\checkmark}\exists_{relative} \gg \mathit{if} \gg \exists_{lawyer}$

Is this predicted by alternative semantics?

Question: selectivity

Two indefinites on an island have the ability to take exceptional scope in different ways outside the island.

- If [a renowned expert on tax law visits a rich relative of mine], I'll inherit a house. $^{\checkmark}\exists_{lawyer} \gg \mathit{if} \gg \exists_{relative}$, $^{\checkmark}\exists_{relative} \gg \mathit{if} \gg \exists_{lawyer}$

Is this predicted by alternative semantics? **NO!!** The grammar *insists* on returning a set of propositions as a meaning for the island.

$$\{\textbf{visits}\ y\ x \mid \textbf{expert}\ x, \textbf{rel}\ y\}$$

Good luck using this set to give one indefinite but not the other scope over the conditional!

Question: binding

Indefinites can be bound into and out of:

- No candidate$_i$ submitted a paper he$_i$ had written.
- A candidate$_i$ submitted her$_i$ best paper.

Is this predicted (allowed) by alternative semantics?

Question: binding

Indefinites can be bound into and out of:

- No candidate$_i$ submitted a paper he$_i$ had written.
- A candidate$_i$ submitted her$_i$ best paper.

Is this predicted (allowed) by alternative semantics? **Well...**

- Alternative semantics is about sets of meanings.
- To talk about pronouns (and, eventually, binding) we need to bring in the assignment-friendly notion of composition, as well.

So, straightaway, there is a question about how to *combine* two enrichments. How should it be done? *Can* it be done in a decent way?

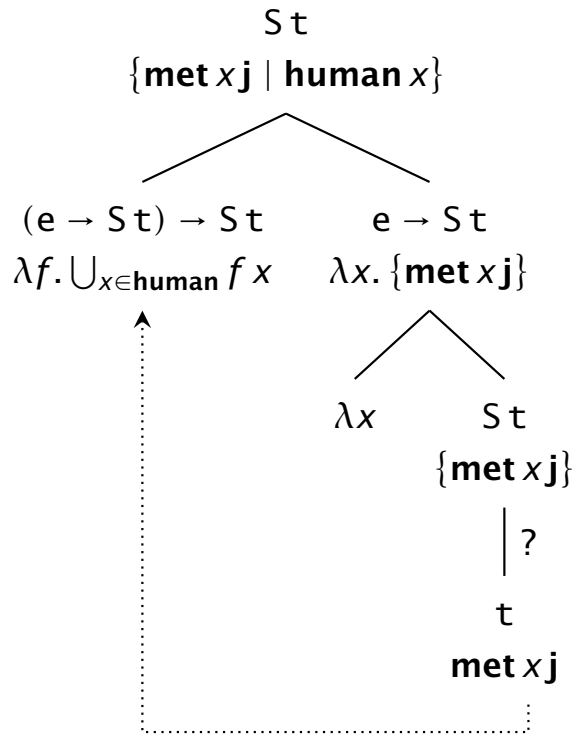
Another approach: Karttunen

Another approach to alternatives was pioneered by Karttunen (1977). Instead of treating wh-words and indefinites as sets of alternatives and then upgrading composition to accommodate them, invoke some compositional magic:

$$? := \lambda p.\, \{p\}$$
$$\textbf{who} := \lambda f.\, \bigcup_{x \in \textbf{human}} f\, x$$

A basic Karttunen-esque derivation

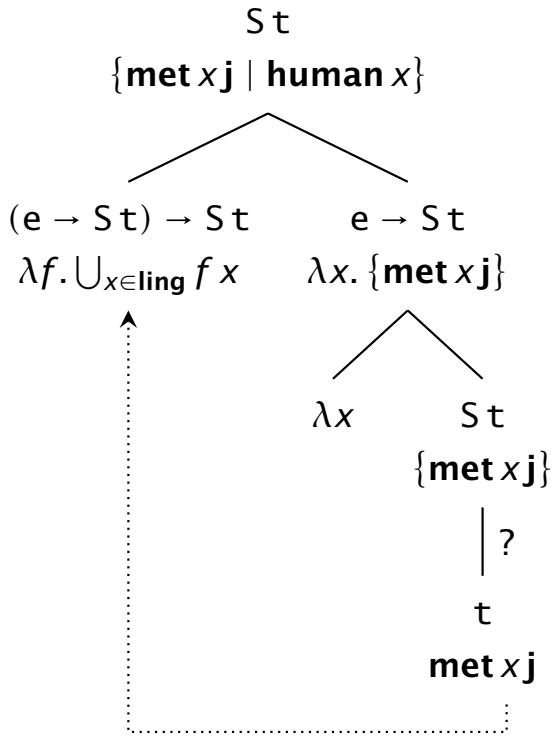
Here, we derive a meaning for *John met who?*



Notice that, just as in the case of quantification, *scope-taking* is a crucial part of the story.

Extending to indefinites (Heim 2014)

Here, we derive a meaning for *John met a linguist.*



The story is exactly the same. Notice, however, that we don't want to commit ourselves to thinking of declarative sentences with indefinites and questions as *precisely* the same sort of object.

Some questions

This is a nice idea, and different in kind from the *enrich-away* strategy we explored earlier.

- Composition remains **Combine** (here, just forward and backward functional application).

But some questions remain:

- What is the relationship between the Karttunen meanings for indefinites and their "standard" quantificational uses?
- Some kind of abstract pattern seemed in evidence for the enrichments we explored before. Can we exploit this to *generalize* Karttunen to other cases?
- And would we want to?

Island effects?

And what about islands? Karttunen's approach crucially relies on scope. Does that mean that adopting it means giving up island-escaping readings for indefinites?

That might be too high a price to pay.

And by the way

Did you notice that island-escaping readings are also characteristic of all the other enrichments we looked at before?

- Every linguist$_i$ would be shocked if [Chomsky cited them$_i$].
- If [the King of France is at the party], John will be upset.
- Which linguist will be offended if [we invite which philosopher]?
- Dr. Svenson only complains if [MARY leaves the lights on].
- If [Mary, who's a linguist, comes to the party], John will be upset.

Well, except for one!

- If [every linguist comes to the party], John will be upset.

Again, scope-taking is the sui generis effect, The odd one out!

Our big questions

So those will be our big questions in this seminar.

- Just what is the relevant abstraction underlying various compositional enrichments?
- Is Karttunen on to something? Can his approach be generalized? Can it be generalized in a way that allows island-escape?
- And what on earth is going on with scope-taking? QR doesn't seem to correspond to the abstraction that underlies enrichments for binding, presupposition, focus, and so on. Is this just a sociological fact? Or is it something deeper?

How we'll approach these questions

We'll look to computer scientists for answers.

- In fact, we'll discover that the Karttunen theory corresponds very directly to an abstraction that computer scientists (well, functional programmers) use *all the time*, to deal with problems *very much like* the ones linguists care about!

Does this seem like a weird thing to do?

Some computer science problems

Defining an arithmetic interpreter

Something a programmer might do is define an arithmetic language:

```
Exp ::= Num Int | Add Exp Exp | Mult Exp Exp | Div Exp Exp
```

So you get expressions like

```
Add (Num 3) (Mult (Num 2) (Num 5))
```

Which evaluates to...?

Data constructors

A note on notation. Things like `Num`, Add, and so on are known as **data constructors**. They may seem odd, but in fact you're already familiar with a data constructor:

$$(\cdot\,,\cdot)$$

Of course, we tend to think of the pair constructor model-theoretically. But in principle, you can think of pairs more abstractly than this. That is, there's a variety of implementations of pairs that "do pairs justice". For example:

$$(a, b) := \lambda f.\, f\, a\, b$$

Can you think of how to get the first and second member of a pair using this representation?

A basic evaluator

```
eval :: Exp → Int

eval (Num x) = ⟦x⟧

eval (Add u v) = eval u + eval v

...
```

Does eval remind you of anything?

It's *really* similar to our linguistic $[\![\cdot]\!]$!

Evaluating our arithmetic formal language is done compositionally, by associating various data constructors with various formal operations (much like type-driven application).

And, in the end, everything is glued together by pure functional application.

Things you might add to this evaluator

Variables, perhaps:

```
eval (Div (Num x) (Num y)) = ???
```

The ability for the thing to fail:

```
eval (Div (Num 1) (Num 0)) = ???
```

Logging! You might want to trace evaluation!

```
eval (Add (Num 1) (Num 2)) = i added 1 and 2 to get 3
```

And you might also wish to add nondeterminism (the ability to do multiple calculations in parallel), or the ability to abort a computation in progress. These also have linguistic analogs (what might they be?).

Implementing a first-order language

```
Var ::= X | Y | Z | ...
```

```
Form ::= Even Var | Odd Var | ... | Eq Var Var | ... |
         Exists Var Form | Neg Form | Conj Form Form |
```

How would you write `eval` for this language? What would be similar to the arithmetic example? What would be different?

Could you also imagine ways that you might wish to extend a first-order language?