

Meaning as computation?

Simon Charlow (Rutgers)

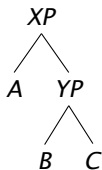
UCSD Ling 230

May 30, 2017

Some semantics

In search of semantics

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.

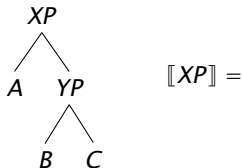
- ▶ This is the folk sense of *compositionality*.

Montague's (Frege's) solution

Define a recursive interpretation function $\llbracket \cdot \rrbracket$, as follows:

$$\llbracket X Y \rrbracket := \mathbf{A} (\llbracket X \rrbracket, \llbracket Y \rrbracket)$$

Then, the interpretation for XP is calculated as follows:

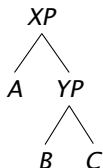


Montague's (Frege's) solution

Define a recursive interpretation function $\llbracket \cdot \rrbracket$, as follows:

$$\llbracket X Y \rrbracket := \mathbf{A} (\llbracket X \rrbracket, \llbracket Y \rrbracket)$$

Then, the interpretation for XP is calculated as follows:

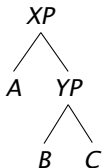

$$\begin{array}{l} XP \\ \swarrow \quad \searrow \\ A \quad YP \\ \quad \swarrow \quad \searrow \\ \quad B \quad C \end{array} \quad \begin{array}{l} \llbracket XP \rrbracket = \mathbf{A} (\llbracket A \rrbracket, \llbracket YP \rrbracket) \\ = \end{array}$$

Montague's (Frege's) solution

Define a recursive interpretation function $\llbracket \cdot \rrbracket$, as follows:

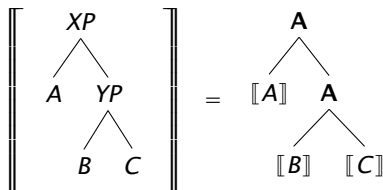
$$\llbracket X Y \rrbracket := \mathbf{A} (\llbracket X \rrbracket, \llbracket Y \rrbracket)$$

Then, the interpretation for XP is calculated as follows:



$$\begin{aligned}\llbracket XP \rrbracket &= \mathbf{A} (\llbracket A \rrbracket, \llbracket YP \rrbracket) \\ &= \mathbf{A} (\llbracket A \rrbracket, \mathbf{A} (\llbracket B \rrbracket, \llbracket C \rrbracket))\end{aligned}$$

Taking a higher view



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

- ▶ This is Montague's (1970) characterization 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 A ?

The 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 from this space):

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

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).]

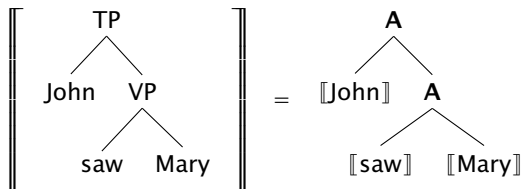
The standard answer, Part 2

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

$$\mathbf{A}(x, y) := \begin{cases} xy & \text{if } x : A \rightarrow B, \text{ and } y : A \quad (\text{FA}) \\ yx & \text{if } x : A, \text{ and } y : A \rightarrow B \quad (\text{BA}) \end{cases}$$

Thus, **A** does forward or backward application, depending on the types of the expressions to be combined (cf. Klein & Sag 1985).

An example



Which ends up reducing to $\text{saw}(\text{Mary})(\text{John})$.

- ▶ John and Mary are the individuals John and Mary, type e .
- ▶ saw is the $f : e \rightarrow e \rightarrow t$ mapping two individuals to t iff they stand in the seeing relation.

Ignorance was bliss

We can do some things w/this semantics, but there's lots we can't handle:

- ▶ Pronouns (and binding), intensionality (and intensional ops)
- ▶ Questions (i.e., as sets of propositions)
- ▶ Focus (and association with focus)
- ▶ Supplemental content (projection)
- ▶ Quantification (and scope)

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

Pronouns and binding

NL has *variable* expressions, which seem to acquire their meanings from the context, or from other expressions in a sentence/discourse:

1. John saw **her**.
2. **Every boy_i** turned in **his_i** homework late.
3. If a **farmer_i** owns a **donkey_j**, **she_i** feeds **it_j** apples.

Currently, we've got no way to talk about the meanings of pronominal expressions. So we'll need to enrich our underlying architecture.

Pronouns: one strategy

Some things depend on the assignment in an essential way:

$$\llbracket \text{her}_i \rrbracket^g := g_i$$

Others meanings are unchanged (the assignment is idle):

$$\llbracket \text{John} \rrbracket^g := \mathbf{j} \quad \llbracket \text{saw} \rrbracket^g := \mathbf{saw}$$

$\llbracket \cdot \rrbracket$ is *upgraded*, to an assignment-friendly $\llbracket \cdot \rrbracket^g$:

$$\llbracket X Y \rrbracket^g := \mathbf{A}(\llbracket X \rrbracket^g, \llbracket Y \rrbracket^g)$$

An equivalent view

Some things depend on the assignment in an essential way:

$$\llbracket \text{her}_i \rrbracket := \lambda g. g_i$$

Others meanings are unchanged (the assignment is idle):

$$\llbracket \text{John} \rrbracket := \lambda g. \mathbf{j} \quad \llbracket \text{saw} \rrbracket := \lambda g. \mathbf{saw}$$

$\llbracket \cdot \rrbracket$ is *upgraded*, to an assignment-friendly $\llbracket \cdot \rrbracket^+$:

$$\llbracket X Y \rrbracket^+ :=$$

An equivalent view

Some things depend on the assignment in an essential way:

$$\llbracket \text{her}_i \rrbracket := \lambda g. g_i$$

Others meanings are unchanged (the assignment is idle):

$$\llbracket \text{John} \rrbracket := \lambda g. \mathbf{j} \quad \llbracket \text{saw} \rrbracket := \lambda g. \mathbf{saw}$$

$\llbracket \cdot \rrbracket$ is *upgraded*, to an assignment-friendly $\llbracket \cdot \rrbracket^+$:

$$\begin{aligned} \llbracket X Y \rrbracket^+ &:= \mathbf{A}^+ (\llbracket X \rrbracket^+, \llbracket Y \rrbracket^+) \\ &:= \end{aligned}$$

An equivalent view

Some things depend on the assignment in an essential way:

$$\llbracket \text{her}_i \rrbracket := \lambda g. g_i$$

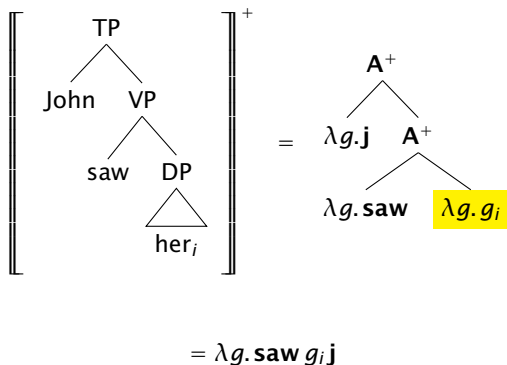
Others meanings are unchanged (the assignment is idle):

$$\llbracket \text{John} \rrbracket := \lambda g. \mathbf{j} \quad \llbracket \text{saw} \rrbracket := \lambda g. \mathbf{saw}$$

$\llbracket \cdot \rrbracket$ is *upgraded*, to an assignment-friendly $\llbracket \cdot \rrbracket^+$:

$$\begin{aligned} \llbracket X Y \rrbracket^+ &:= \mathbf{A}^+ (\llbracket X \rrbracket^+, \llbracket Y \rrbracket^+) \\ &:= \lambda g. \mathbf{A} (\llbracket X \rrbracket^+ g, \llbracket Y \rrbracket^+ g) \end{aligned}$$

Example derivation



Binding

Chierchia & McConnell-Ginet (2000) style:

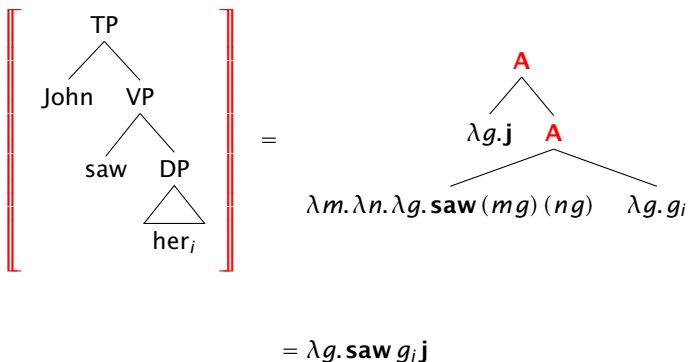
$$\llbracket Op_i X \rrbracket^g := \llbracket Op \rrbracket^g (\lambda x. \llbracket X \rrbracket^{g[i \rightarrow x]})$$

Heim & Kratzer (1998) style:

$$\llbracket \lambda_i X \rrbracket^g := \lambda x. \llbracket X \rrbracket^{g[i \rightarrow x]}$$

These rules are given *syncategorematically*: $\llbracket Op_i / \lambda_i X \rrbracket^g$ is not stated in terms of $\llbracket Op_i / \lambda_i \rrbracket^g$ and $\llbracket X \rrbracket^g$. Why not?

Going lexicalist (e.g., Sternefeld 1998, 2001)



Duplicated work

All the approaches we've seen involve a degree of “generalization to the worst case”: we lexically duplicate some predictable patterns:

$$\llbracket \text{John} \rrbracket^+ := \lambda g. \mathbf{j} \quad \llbracket \text{saw} \rrbracket^+ := \lambda g. \mathbf{saw}$$

Or cf. the lexicalists' lexical entries:

$$\llbracket \text{saw} \rrbracket = \lambda m. \lambda n. \lambda g. \mathbf{saw} (mg) (ng) \quad \llbracket \text{ate} \rrbracket = \lambda m. \lambda n. \lambda g. \mathbf{ate} (mg) (ng)$$

Abstracting out the essence

What's essential

Handling pronouns compositionally requires three pieces:

- ▶ Relatively fancy meanings for pronominal expressions
- ▶ A common way of dealing with boring (non-pronominal) expressions
- ▶ A common way of dealing with fancy meaning combination

The first is non-negotiable. The other two can be packaged up in various ways, with varying degrees of lexical duplication.

What would things look like if we kept the lexicon *maximally simple*?

Two functions

A function for turning boring things into maximally boring fancy things:

Two functions

A function for turning boring things into maximally boring fancy things:

$$\eta x := \lambda g. x$$

Along with a function for composing two fancy things to yield a third:

$$+ F :=$$

Two functions

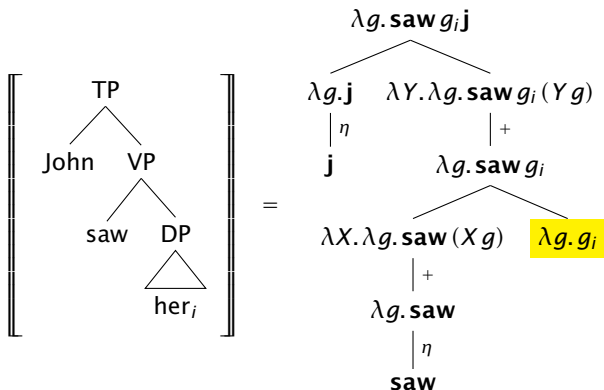
A function for turning boring things into maximally boring fancy things:

$$\eta x := \lambda g. x$$

Along with a function for composing two fancy things to yield a third:

$$+ F := \lambda X. \lambda g. F g (X g)$$

Example derivation



[Binary-branching nodes compose uniformly via A!]

Questions: intuition

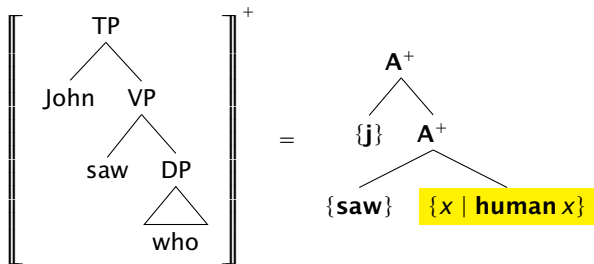
A common approach to question semantics (Hamblin 1973, Karttunen 1977) treats questions as denoting *sets of their possible answers*:

$$\llbracket \text{who did John see?} \rrbracket = \{ \text{saw } xj \mid \text{human } x \}$$

And a common approach to deriving sets of answers is to begin by treating $\llbracket \text{who} \rrbracket$ as a set of *alternatives*, $\{x \mid \text{human } x\}$.

Like pronouns, we have an immediate compositional challenge: how to compose sets of alternatives (cf. assignment-dependent meanings) to yield bigger sets of alternatives (cf. assignment-dependent meanings)?

Alternative semantics (Hamblin 1973)



$$[[X\ Y]]^+ := \{A(x, y) \mid x \in [[X]]^+, y \in [[Y]]^+\}$$

Result: $\{saw\ x\ j \mid \mathbf{human\ } x\}$

Abstracting out the common pattern

Could we pull a similar trick as we did with pronouns? Finding two functions that help us do our job without complicating the lexicon?

A function for turning boring things into maximally boring fancy things:

$$\eta x :=$$

Abstracting out the common pattern

Could we pull a similar trick as we did with pronouns? Finding two functions that help us do our job without complicating the lexicon?

A function for turning boring things into maximally boring fancy things:

$$\eta x := \{x\}$$

Along with a function for composing two fancy things to yield a third:

$$+ F :=$$

Abstracting out the common pattern

Could we pull a similar trick as we did with pronouns? Finding two functions that help us do our job without complicating the lexicon?

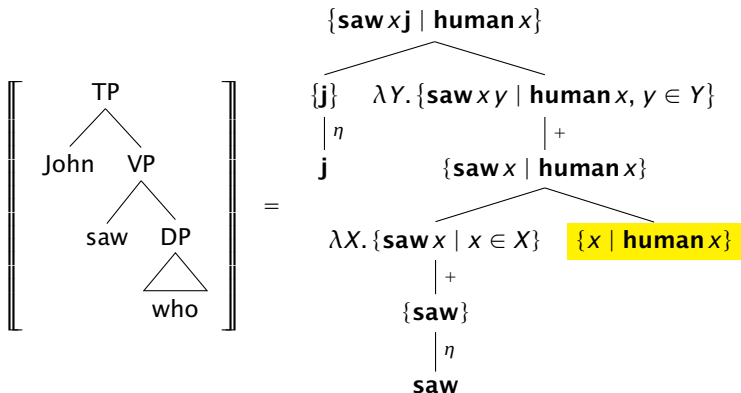
A function for turning boring things into maximally boring fancy things:

$$\eta x := \{x\}$$

Along with a function for composing two fancy things to yield a third:

$$+F := \lambda X. \{f x \mid f \in F, x \in X\}$$

Example derivation



[Binary-branching nodes compose uniformly via A!]

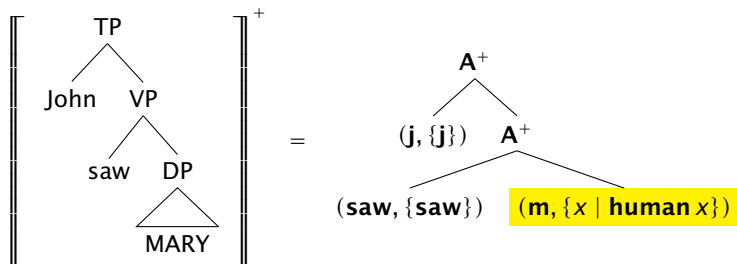
Focus

In *association with focus*, we consider an utterance, alongside other things its speaker might have said:

1. I only introduced JOHN to Mary.

≈ I introduced John to Mary, and I didn't introduce anyone else to Mary.

Example



$$= (\text{saw } mj, \{\text{saw } xj \mid \text{human } x\})$$

$$[[X Y]^+ := (\mathbf{A}([X]_1^+, [Y]_1^+), \{\mathbf{A}(x, y) \mid x \in [X]_2^+, y \in [Y]_2^+\})$$

Finding our two functions

A simple injection function:

$$\eta x := (x, \{x\})$$

Alongside a notion of fancy application:

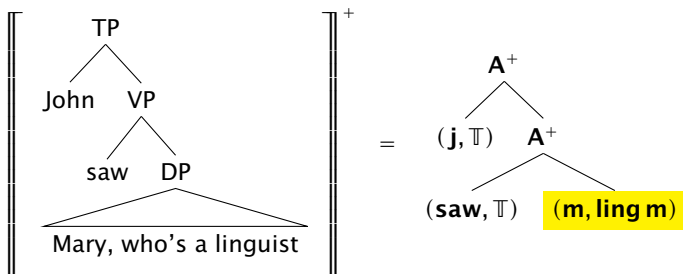
$$+ F := \lambda X. (F_1 X_1, \{f x \mid F \in F_2, x \in X_2\})$$

Supplemental content

1. John met Mary, who's a linguist.
2. John didn't meet Mary, who's a linguist.

Supplemental content is somehow separate from everything else.

Example

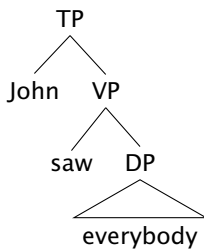


= (saw m j, ling m)

$$[X Y]^+ := (A ([X]_1, [Y]_1), [X]_2 \wedge [Y]_2)$$

What functions might underlie this pattern?

Scope

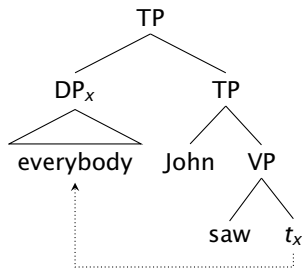


The problem of scope-taking: some expressions need access to more than their immediate semantic context:

everybody ($\lambda x. \text{saw } xj$)

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.)

Even scope-taking yields

$$\eta x := \lambda k. k x$$

$$+ F := \lambda X. \lambda k. F (\lambda f. X (\lambda x. k (f x)))$$

Take a deep breath.

You're in the presence of **continuations** (Barker 2002).

Applicative functors

FUNCTIONAL PEARL

Applicative programming with effects

CONOR MCBRIDE¹

University of Nottingham

ROSS PATERSON

City University, London

Abstract

In this article, we introduce `Applicative` functors – an abstract characterisation of an applicative style of effectful programming, weaker than `Monads` and hence more widespread. Indeed, it is the ubiquity of this programming pattern that drew us to the abstraction. We retrace our steps in this article, introducing the applicative pattern by diverse examples, then abstracting it to define the `Applicative` type class and introducing a bracket notation that interprets the normal application syntax in the idiom of an `Applicative` functor. Furthermore, we develop the properties of applicative functors and the generic operations they support. We close by identifying the categorical structure of applicative functors and examining their relationship both with `Monads` and with `Arrows`.

A connection

The abstraction talked about in this recent, influential computer science paper, is *precisely* the same abstraction we've been using to re-orient our grammars to handle enriched notions of composition!

Our abstractions are all **applicative functors**!

Should we be surprised by this?

Functional languages

Functional programming languages are, just like NL grammars in the Frege/Montague tradition, built on functions and arguments.

Just like NL semanticists, functional programmers frequently find themselves hungering for *effects*, and ways to systematically express concepts that lie outside the core features of their language.

Some common effects

- ▶ Environment (valuing variables in a global namespace)
- ▶ Nondeterminism (carrying out multiple computations in parallel)
- ▶ Pointed nondeterminism (flagging a particular value as central)
- ▶ Logging (keeping a side-log of execution-incidental info)
- ▶ Control (aborting a computation, jumping around inside a program)

These all correspond in a fairly direct way to things that are useful for doing natural language semantics!

Other abstractions?

McBride & Paterson (2008) reference some other abstractions — monads and arrows — as possible alternatives to applicative functors.

Are these abstractions useful for semantics? Are there any reasons to use them, or even, perhaps, to *prefer* one abstraction to another?

And are there arguments for these kinds of approaches, over standard ones? (Come see me Thursday to find out!)

And by the way

Did you notice that *island-escaping readings* are characteristic of (almost) all the other enrichments we've considered today?

- ▶ If [a rich relative of mine dies] I'll inherit a house.
- ▶ Every linguist_{*i*} would be shocked if [Chomsky cited them_{*i*}].
- ▶ 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!

- Barker, Chris. 2002. Continuations and the nature of quantification. *Natural Language Semantics* 10(3). 211–242. <http://dx.doi.org/10.1023/A:1022183511876>.
- Chierchia, Gennaro & Sally McConnell-Ginet. 2000. *Meaning and grammar: An introduction to semantics*. Second edition. Cambridge, MA: MIT Press.
- Hamblin, C. L. 1973. Questions in Montague English. *Foundations of Language* 10(1). 41–53.
- Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. Oxford: Blackwell.
- Karttunen, Lauri. 1977. Syntax and semantics of questions. *Linguistics and Philosophy* 1(1). 3–44. <http://dx.doi.org/10.1007/BF00351935>.
- Klein, Ewan & Ivan A. Sag. 1985. Type-driven translation. *Linguistics and Philosophy* 8(2). 163–201. <http://dx.doi.org/10.1007/BF00632365>.
- McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. <http://dx.doi.org/10.1017/S0956796807006326>.
- Montague, Richard. 1970. Universal grammar. *Theoria* 36(3). 373–398. <http://dx.doi.org/10.1111/j.1755-2567.1970.tb00434.x>.
- Sternefeld, Wolfgang. 1998. *The semantics of reconstruction and connectivity*. Arbeitspapier 97, SFB 340. Universität Tübingen & Universität Stuttgart, Germany.
- Sternefeld, Wolfgang. 2001. Semantic vs. syntactic reconstruction. In Christian Rohrer, Antje Roßdeutscher & Hans Kamp (eds.), *Linguistic Form and its Computation*, 145–182. Stanford: CSLI Publications.