# A modular theory of pronouns and binding

Simon Charlow

Rutgers, The State University of New Jersey, New Brunswick NJ 08901, USA
simon.charlow@rutgers.edu

**Abstract.** I sketch a modular treatment of pronouns using applicative functors, monads, and then applicative functors (again!). This approach dissolves theoretical issues typical of standard accounts, and extends their empirical coverage to paycheck pronouns and binding reconstruction.

**Keywords:** paycheck pronouns, binding reconstruction, applicative functors, monads, variable-free semantics

## 1 Overview

Since Shan's (2002) pioneering work, a number of researchers have argued that MONADS offer a flexible, robust compositional interface for expressions that denote in "enriched spaces" (e.g., Giorgolo & Asudeh 2012, Unger 2012, Charlow 2014, 2017). This paper argues that a monadic treatment of pronouns and assignment-sensitivity has a number of theoretical and empirical benefits, including (i) a maximally simple lexicon and a fully categorematic treatment of abstraction; (ii) centrally, immediate analyses of PAYCHECK PRONOUNS and BINDING RECONSTRUCTION with a unitary, simple semantics for pronouns and traces: $[\![\text{she}_i]\!] = [\![\text{t}_i]\!] = \lambda g.\, g_i$. The treatment involves abstracting out the two functions that underlie standard treatments of assignment-friendly composition — yielding a so-called APPLICATIVE FUNCTOR (McBride & Paterson 2008) — and then adding a third function to deal with 'higher-order' variables, yielding a monad.

Two developments of the basic idea are briefly explored. First, I argue that a mere applicative functor turns out to be sufficient after all, if we (a) adopt a more type-theoretically conservative treatment of assignments than is standard, and (b) countenance sentence meanings that depend on multiple assignments. Second, I demonstrate an equivalence of the resulting theory with a VARIABLE-FREE SEMANTICS (cf., e.g., Jacobson 1999), one which extends both to the combinatory apparatus underwriting composition, and to the resulting semantic values.

## 2 The standard theory, and its discontents

### 2.1 Adding assignments

When we do model-theoretic semantics, we characterize the kinds of meanings expressions can have and specify a procedure for building complex meanings from

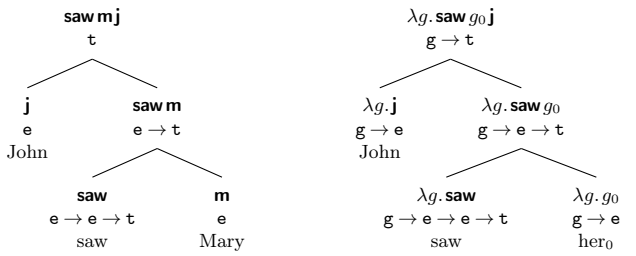


**Fig. 1.** Left: deriving a meaning for *John saw Mary* in a grammar based on $\tau$ and $[\![\cdot]\!]$. Right: deriving a meaning for *John saw her$_0$* in a grammar based on $\tau^+$ and $[\![\cdot]\!]$.

smaller pieces. A particularly simple model in the Fregean vein is given below: (1) says that meanings can either be entities (type $\mathtt{e}$), truth values (type $\mathtt{t}$), or functions from meanings to meanings; (2) says that the meaning of a binary branching node is gotten by doing (type-driven) functional application on the denotations of its daughters. See Figure 1 (left) for an example derivation.

(1) $\tau ::= \mathtt{e} \mid \mathtt{t} \mid \tau \to \tau$

(2) $[\![\alpha\, \beta]\!] \coloneqq [\![\alpha]\!]\,[\![\beta]\!]$ or $[\![\beta]\!]\,[\![\alpha]\!]$, whichever's defined

This baseline theory can be used to profitably theorize about a large, nontrivial fragment of natural language. But there's a lot it doesn't handle. My focus in this paper is on *variable* expressions — that is, on free and bound pronouns:

(3) John saw her.

(4) Every philosopher$_i$ thinks they$_i$'re a genius.

The simple model given by (1) and (2) seems ill-suited to such cases: no concrete member of type $\mathtt{e}$ is a suitable candidate for the denotation of a free prounoun (whose reference shifts with the context of utterance) or a bound pronoun (whose reference shifts as the binder plows through its domain).

The standard Tarskian treatment of pronouns (e.g., Heim & Kratzer 1998) is meant to remedy this. It's got two pieces. First, meanings are uniformly conceived of as determined relative to some way of valuing free pronouns — commonly known as an assignment function, as indicated in (5). So both pronominal and nonprominal expressions have denotations of the form '$\lambda g$...' (here and throughout I represent assignment-dependence explicitly via a $\lambda$-abstract). Second, the bare-bones Fregean interpretation function $[\![\cdot]\!]$ is re-engineered as $[\![\cdot]\!]$, a function that composes assignment-dependent meanings to yield new assignment-dependent meanings, as in (6) ($[\![\cdot]\!]$ is akin to Heim & Kratzer's $[\![\cdot]\!]^g$). A derivation of *John saw her$_0$* using these pieces is depicted in Figure 1, right.

(5) $\tau^+ ::= \mathtt{g} \to \tau$

(6) $[\![\alpha\, \beta]\!] \coloneqq \lambda g.\, [\![\alpha]\!]\, g\, ([\![\beta]\!]\, g)$ or $[\![\beta]\!]\, g\, ([\![\alpha]\!]\, g)$, whichever's defined

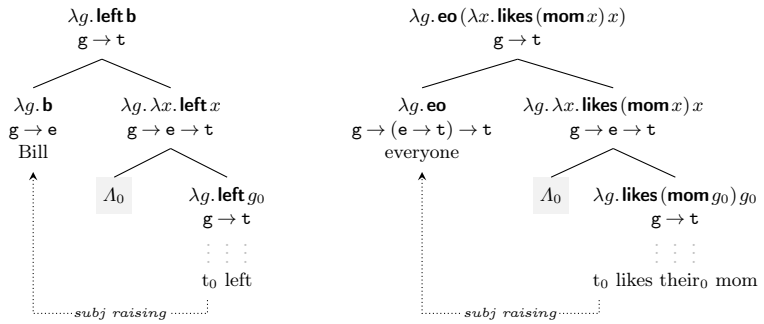


**Fig. 2.** Derivations requiring a syncategorematic treatment of $\Lambda_0$ in the presence of $[\![\cdot]\!]$.

### 2.2 Conceptual and empirical issues

The theoretical and empirical discontents of the standard approach are well known. On the theoretical side, it requires lexically generalizing to the worst case: everything is treated as assignment-sensitive, including expressions that are not usually taken to depend on assignments in any meaningful way (e.g., proper nouns, verbs, and so on). For example, the derivation in Figure 1 (right) requires trivially assignment-dependent lexical entries for *John* and *saw* — $[\![\cdot]\!]$ demands that its inputs be assignment-relative, and so the lexicon must oblige.[1]

Also, because the grammar insists on interpreting binary-branching nodes by passing an input assignment to both branches, any behavior deviating from this norm (e.g., abstraction) requires a bespoke syncategorematic rule. Consider the trees in Figure 2, in which we aim to bind a raised subject's trace (left), and a co-indexed pronoun (right). In both cases, $\Lambda_0$'s sister must be evaluated at *shifted* assignments $g^{0\to x}$ anchored to a new functional parameter $\lambda x$. But no possible denotation for $\Lambda_0$ can secure this result: once $[\![\cdot]\!]$ passes the input assignment to $\Lambda_0$'s sister, the latter can't be *un*saturated — the jig is up. Getting the right result for such cases thus requires a syncategorematic rule: $[\![\Lambda_0\,\alpha]\!] \coloneqq \lambda g.\,\lambda x.\,[\![\alpha]\!]\,g^{0\to x}$.

On the empirical side, the standard account doesn't handle paycheck constructions (Cooper 1979, Engdahl 1986) — e.g., the reading of (7) saying Bill hates *Bill's* mom or the reading of (8) saying every linguist saved *their* paycheck.[2]

(7) John$_i$ likes [his$_i$ mom]$_j$. But Bill$_k$ hates her$_j$.

(8) Every philosopher$_i$ spent [their$_i$ paycheck]$_j$. Every linguist$_k$ saved it$_j$.

There are two related puzzles here. First: in (7) and (8), the second, PAYCHECK PRONOUN is quite clearly anaphoric to the co-indexed expression in the previous

[1] A related point about lexical complication can be made about theories built on simple functional application which task their lexical entries with doing all the assignment management (e.g., Sternefeld 1998, 2001, Kobele 2010).

[2] This is, of course, not to say that the standard account is incompatible with paycheck readings, only that it does not generate them out of the box and therefore requires additional stipulations (Engdahl 1986). See Jacobson (2000) for discussion.

sentence. Yet despite this anaphoric relationship, the two expressions somehow come to have different meanings (e.g., John's mom vs. Bill's mom). Second: the lexical semantics of pronouns is given schematically as $[\![\text{pro}_i]\!] \coloneqq \lambda g.\, g_i$. Even if *Bill* or *every linguist* triggers an assignment shift via a $\Lambda_k$ operator, this shift should have no effect whatsoever on the interpretations of the paycheck pronouns, which bear a distinct index $j$. So on standard assumptions about pronouns, how they could ever come to be 'bound into' (as paychecks seem to be) is mysterious.

Nor does the standard account generate cases of BINDING RECONSTRUCTION, as in the indicated readings of (9) and (10). The reason is simple: Predicate Abstraction passes modified assignments *down the tree*, and so binding invariably requires (LF) c-command (e.g., Sternefeld 1998, 2001, Barker 2012).

(9) [His$_i$ mom]$_j$, every boy$_i$ likes $t_j$.

(10) [Unless he$_i$'s been a bandit]$_j$, no man$_i$ can be an officer $t_j$.

We might hope to rescue these cases by scoping the quantifier over the fronted pronoun. But this should trigger a Weak Crossover violation, as in the ungrammatical **his$_i$ mom likes every boy$_i$*. Indeed, the distinctive feature of binding reconstruction is that the trace of the overtly moved expression occupies a *lower* (roughly: further right) position than the quantifier. In other words, binding reconstruction requires the bound-into expression to 'originate' lower than the binder. A theory of binding reconstruction should capture this fact.

## 3 Getting modular

### 3.1 Abstracting out the essence of the standard account

An alternative approach to assignments is to simply abstract out and modularize the core features of the standard account, as in (11) and (12). Instead of treating the lexical-semantic values of non-pronominals as trivially dependent on an assignment, we'll invoke a function $\rho$ which turns any $x$ into a constant function from assignments into $x$. Instead of taking on $[\![\cdot]\!]$ wholesale, we'll help ourselves to a function $\circledast$ which perform assignment-friendly function application on demand.

(11) $\rho\, x \coloneqq \lambda g.\, x$ $\qquad\qquad \rho :: a \to \mathsf{g} \to a$

(12) $m \circledast n \coloneqq \lambda g.\, m\, g\, (n\, g)$ $\qquad\qquad \circledast :: (\mathsf{g} \to a \to b) \to (\mathsf{g} \to a) \to \mathsf{g} \to b$

Derivations of *she$_0$ left* and *John saw her$_0$* using these pieces are provided in Figure 3. The principles guiding the construction of these derivations are quite simple. Wherever the standard account appeals to a trivially assignment-dependent lexical entry, we instead invoke $\rho$. Wherever the standard account appeals to $[\![\cdot]\!]$, we instead invoke $\circledast$ (applied to the assignment-sensitive function).

### 3.2 Conceptual issues dissolved

The $\rho/\circledast$ approach to compositionally managing assignment-sensitivity immediately dissolves the theoretical baggage associated with the standard account.

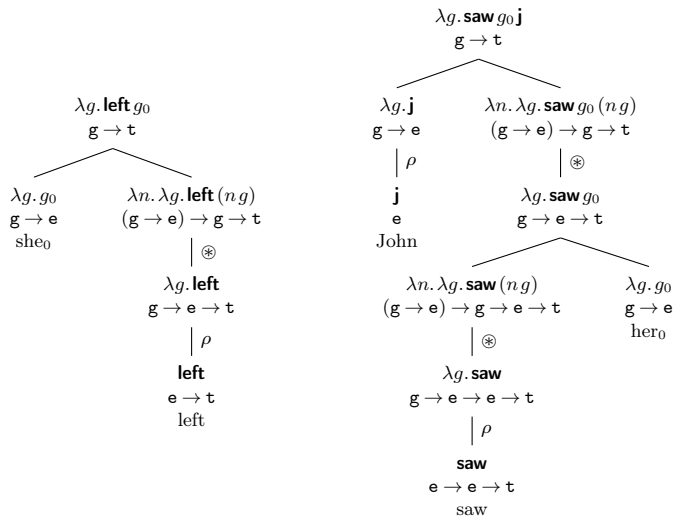


**Fig. 3.** Assignment-sensitive composition using $\rho$, $\circledast$, and $[\![\cdot]\!]$, cf. Figure 1 (right).

First, $\rho$ lets us keep the lexicon maximally simple: anything that isn't really assignment-dependent can be entered into the lexicon as such.

Second, $\circledast$ liberates us from the yoke of $[\![\cdot]\!]$, which allows us to categorematically define operations relevant for binding, as in (13). Because the grammar is oriented around simple functional application, and assignment-friendly functional application ($\circledast$) is conjured only as needed, there's no grammatical default about how assignment functions get passed around (as there is with $[\![\cdot]\!]$), and no need for a syncategorematic rule subverting a grammatical default that doesn't exist.

(13) $\Lambda_i \coloneqq \lambda f.\, \lambda g.\, \lambda x.\, f\, g^{i \to x}$

Derivations using $\Lambda_i$ to generate binding of a trace and concomitant binding of a pronoun are given in Figure 4. (I've elected to notate $\Lambda_i$ as a unary compositional rule rather than as a lexical item, but nothing here hinges on this choice.)

### 3.3 On applicatives

When we abstract out $\rho$ and $\circledast$ in this way, we're in the presence of something known to computer scientists and functional programmers as an APPLICATIVE FUNCTOR (McBride & Paterson 2008). Essentially, an applicative functor characterizes an enriched type-space that supports some correspondingly enriched notion of functional application. The relevant enrichment for our purposes is assignment-sensitivity, and the corresponding enrichment of functional application is, naturally enough, assignment-friendly functional application.

A little more formally, an applicative functor is a type constructor $F$ associated with two functions $\rho :: a \to F\, a$ and $\circledast :: F\,(a \to b) \to F\, a \to F\, b$. These two functions

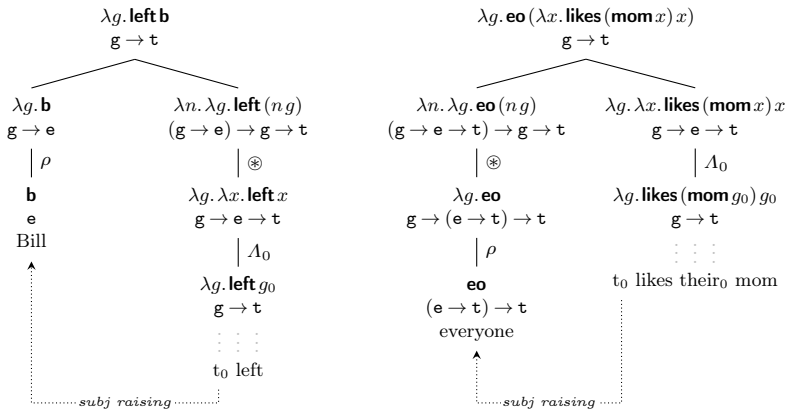


**Fig. 4.** Binding derivations using a categorematic $\Lambda_0$ alongside $\rho$, $\circledast$, and $[\![\cdot]\!]$, cf. Figure 2.

are required to satisfy the four laws below, which together guarantee that $\circledast$ embodies an enriched notion of functional application, and that $\rho$ does nothing more than trivially inject values into the enriched type-space characterized by $F$.

| **Homomorphism** | **Identity** |
|---|---|
| $\rho f \circledast \rho x = \rho(f x)$ | $\rho(\lambda x.\, x) \circledast v = v$ |
| **Interchange** | **Composition** |
| $\rho(\lambda f.\, f x) \circledast u = u \circledast \rho x$ | $\rho(\circ) \circledast u \circledast v \circledast w = u \circledast (v \circledast w)$ |

Our applicative functor is given by $\mathtt{G}a ::= \mathtt{g} \to a$, with $\rho$ and $\circledast$ as defined in (11) and (12). It's straightforward to check that these $\rho$ and $\circledast$ have types of the right shape, and that they satisfy the four applicative functor laws.

Applicative functors can be factored out of a great deal of existing semantic theory. I'll briefly mention two examples. Alternative semantics of the Hamblin (1973) variety can be re-stated using an applicative functor for sets, such that $\mathtt{S}a ::= a \to \mathtt{t}$, with $\rho$ and $\circledast$ as defined in (14) and (15).

(14) $\rho x \coloneqq \{x\}$ $\qquad \rho :: a \to \mathtt{S}\,a$

(15) $m \circledast n \coloneqq \{f x \mid f \in m, x \in n\}$ $\qquad \circledast :: \mathtt{S}\,(a \to b) \to \mathtt{S}\,a \to \mathtt{S}\,b$

Likewise, the continuations-based analyses of Shan & Barker 2006, Barker & Shan 2014 are built on two combinators ('**Lift**' and '**Scope**') that directly instantiate the applicative functor for *continuations*, such that $\mathtt{C}_r\,a ::= (a \to r) \to r$, and:

(16) $\rho x \coloneqq \lambda\kappa.\, \kappa x$ $\qquad \rho :: a \to \mathtt{C}_r\,a$

(17) $m \circledast n \coloneqq \lambda\kappa.\, m\,(\lambda f.\, n\,(\lambda x.\, \kappa\,(f x)))$ $\qquad \circledast :: \mathtt{C}_r\,(a \to b) \to \mathtt{C}_r\,a \to \mathtt{C}_r\,b$

Finally, as emphasized by McBride & Paterson (2008), Kiselyov (2015), applicative functors enjoy an important property: they are closed under composition. Given any two applicative type constructors $F$ and $G$, both $F \circ G$ and $G \circ F$

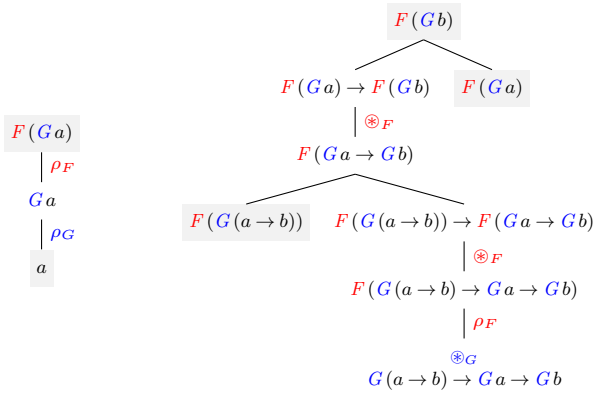


**Fig. 5.** Composing two applicative functors $F$ and $G$.

are applicative as well. The general recipe for assembling composite $\rho$ and $\circledast$ operations from $F$ and $G$'s applicative components is given in Figure 5. For example, $\mathtt{G} \circ \mathtt{S}$ yields assignment-dependent sets of alternatives, with $\rho x = \lambda g.\{x\}$ and $m \circledast n = \lambda g.\{f\,x \mid f \in m\,g, x \in n\,g\}$ (cf. Kratzer & Shimoyama 2002, Charlow 2014, 2017). $\mathtt{S} \circ \mathtt{G}$ gives alternative assignment-dependent meanings, with $\rho x = \{\lambda g.x\}$ and $m \circledast n = \{\lambda g.f\,g\,(x\,g) \mid f \in m, x \in n\}$ (cf. Poesio 1996, Romero & Novel 2013). And composing $\mathtt{G}$ with *itself* — taking $\mathtt{G} \circ \mathtt{G}$ — yields an applicative for double-assignment-relativity: $\rho x = \lambda g.\lambda h.x$ and $m \circledast n = \lambda g.\lambda h.m\,g\,h\,(n\,g\,h)$. This latter composite applicative makes an appearance in Sections 5 and 6.[3]

The 'compositionality' of applicative functors is significant in part because it guarantees that the applicative bits of grammar can be theorized about separately and modularly. Any set of potentially disparate analyses relying on applicatives automatically generates a 'composite' analysis in terms of composed applicatives.

## 4 Getting higher-order

### 4.1 Higher-order meanings

Taking stock, applicative functors are a robust way to add assignment-sensitivity to a baseline grammar, allowing pronouns and traces to interact with expressions that are not inherently assignment-sensitive. This approach dissolves the theoretical baggage inherent in the standard, one-size-fits-all account.

But it would appear that $\rho$ and $\circledast$ have nothing special to say about binding reconstruction or paycheck pronouns (but see Section 5). Intuitively, both phenomena are *higher-order*, in that the referent anaphorically retrieved by the paycheck pronoun or the topicalized expression's trace is an 'intension', rather than an 'extension' (cf. Groenendijk & Stokhof 1990, Sternefeld 1998, 2001, Hardt

[3] Notably, the continuations-based analysis of inverse scope (e.g., Shan & Barker 2006) involves composing $\mathtt{C}_r$ with itself.

1999, Kennedy 2014). In, e.g., (7), the paycheck pronoun is anaphoric to $his_i$ *mom*, but it doesn't thereby denote the extension of $his_i$ *mom* (which is, after all, John's mom). In, e.g., (9), the trace is anaphoric to $his_i$ *mom*, but not to its extension (i.e., the mother of the contextually furnished value for the index $i$).

When assignments (rather than worlds) are our indices of evaluation, the intension of a phrase like $his_0$ *mom* is $\lambda g.\,\mathbf{mom}\,g_0$ (type $\mathtt{g} \to \mathtt{e}$). Its corresponding extension at an assignment mapping 0 to, e.g., $\mathbf{j}$ is $\mathbf{mom}\,\mathbf{j}$ (type $\mathtt{e}$). Similarly, pronouns *anaphoric to* extensions deliver an individual after being served an assignment function; their type is $\mathtt{g} \to \mathtt{e}$. Pronouns anaphoric to *intensions* deliver an intension after being served an assignment; their type is therefore $\mathtt{g} \to \mathtt{g} \to \mathtt{e}$.

There is thus reason to believe that pronouns and traces can have among their types $\mathtt{g} \to \mathtt{e}$ and $\mathtt{g} \to \mathtt{g} \to \mathtt{e}$. Going whole hog, we assume that pronouns and traces are polymorphic, with the recursive type given in (18). Simply put, a pronoun (or trace) is an expression that denotes an individual after being supplied some number of assignment functions.

(18) $\mathtt{pro} ::= \mathtt{g} \to \mathtt{e} \mid \mathtt{g} \to \mathtt{pro}$

Two points. First, the generalized type in (18) doesn't mean pronouns or traces are ambiguous. Indeed, they're assigned a unified (schematic) lexical semantics: $[\![\mathrm{pro}_i]\!] \coloneqq \lambda g.\,g_i$. Second, it should be noted that reasons already exist to treat DP traces as polymorphic along a different dimension: instances of SCOPE RECONSTRUCTION, as in the $\neg \gg \forall$ reading of *everyone didn't pass*, can be analyzed by assuming that traces of overt DP movement (here, subject raising) can be of type $\mathtt{g} \to (\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$ in addition to type $\mathtt{g} \to \mathtt{e}$ (e.g., von Fintel & Heim 2011: 94).

### 4.2 Extending the grammar with a higher-order combinator

So our toolkit now includes a generalized type for pronouns and traces (though a constant lexical semantics). The question now becomes how higher-order pronouns and traces should be folded into derivations.

An obvious option is to invoke a 'flattener' function which turns a higher-order pronoun (type $\mathtt{g} \to \mathtt{g} \to \mathtt{e}$) into something with the same type as a garden-variety pronoun ($\mathtt{g} \to \mathtt{e}$), and then to simply run our derivations as before. See (19) for the definition of the flattener, which we'll write '$\mu$'. Its job is to take an expression $m$ that's anaphoric to an *intension*, and then obtain an *extension* by evaluating the anaphorically retrieved intension $m\,g$ once more against $g$.[4]

(19) $\mu m \coloneqq \lambda g.\,m\,g\,g$ $\qquad\qquad$ $\mu :: (\mathtt{g} \to \mathtt{g} \to a) \to \mathtt{g} \to a$

And that is actually all there is to it! Derivations using $\rho$, $\circledast$, and $\mu$ (along with $[\![\cdot]\!]$ and $\Lambda_i$) to derive a paycheck reading and binding reconstruction (respectively)

[4] A referee wonders whether $\mu$ is not already entailed by $\rho$ and $\circledast$: (untyped versions of) these functions correspond to Combinatory Logic's $\mathbf{K}$ and $\mathbf{S}$ combinators, which are Turing-complete. In particular, in an untyped setting, $\mu$ (aka $\mathbf{W}$) is equivalent to $\mathbf{SS}(\mathbf{SK})$. Importantly, though, this only holds of *untyped* systems. E.g., given any concrete type for the 'environment' argument (here: $\mathtt{g}$), $\mathbf{SS}$ isn't defined.

are given in Figure 6.[5] Notice that the pronoun in the paycheck derivation and the trace in the binding reconstruction derivation have the same sort of semantics they've had all along, but that their types are now higher-order: $\mathsf{g} \to \mathsf{g} \to \mathsf{e}$. These higher-order meanings are converted to garden-variety denotations of type $\mathsf{g} \to \mathsf{e}$ via $\mu$. The rest is a mundane series of invocations of $\rho$, $\circledast$, $\Lambda_i$, and $[\![\cdot]\!]$.

In the paycheck derivation, the meaning computed for the sentence expects the incoming assignment to furnish an intensional value (type $\mathsf{g} \to \mathsf{e}$) for the index 1. In the context of example (7), there's a natural candidate at hand: the meaning of *his*$_0$ *mom* — i.e., $\lambda g.\,\mathbf{mom}\,g_0$! Supposing that the contextually given input assignment $\mathbf{g}$ maps 1 to this value, we observe the following routine series of $\lambda$-theoretic equivalences, resulting in the desired paycheck truth-conditions:

$$\begin{aligned}
(\lambda g.\,\mathbf{hates}\,(g_1\,g^{0\to\mathbf{b}})\,\mathbf{b})\,\mathbf{g} &= \mathbf{hates}\,(\mathbf{g}_1\,\mathbf{g}^{0\to\mathbf{b}})\,\mathbf{b} && \beta\\
&= \mathbf{hates}\,((\lambda g.\,\mathbf{mom}\,g_0)\,\mathbf{g}^{0\to\mathbf{b}})\,\mathbf{b} && \equiv\\
&= \mathbf{hates}\,(\mathbf{mom}\,(\mathbf{g}^{0\to\mathbf{b}})_0)\,\mathbf{b} && \beta\\
&= \mathbf{hates}\,(\mathbf{mom}\,\mathbf{b})\,\mathbf{b} && \equiv
\end{aligned}$$

The binding reconstruction derivation works along essentially identical lines. The only material difference here is that the intension picked up by the higher-order variable (here, the trace) is fixed by the compositional semantics via in-scope binding by $\Lambda_1$, and not by the pragmatics or (pending suitable elaborations, cf. Charlow 2014) via dynamic binding, as happens in the paycheck case. Notice in particular that this analysis of binding reconstruction doesn't require the quantifier to scope over a pronoun it isn't higher than (i.e., to the left of) on the surface. Given the Weak Crossover facts, this is exactly as desired.

### 4.3 On monads

We observed that $\rho$ and $\circledast$ form an applicative functor. And it turns out that $\rho$, $\circledast$, and $\mu$ likewise correspond to a construct familiar to functional programmers — namely, a MONAD (sometimes termed the Environment or Reader monad, cf. Wadler 1995, Shan 2002) — though the presentation here, in terms of an applicative functor plus $\mu$, seems to be novel, at least in the linguistics literature.

In functional programming contexts, monads are usually defined as a type constructor $T$ with two operations $\eta :: a \to T\,a$ and $\gg\!= :: T\,a \to (a \to T\,b) \to T\,b$, satisfying three laws (see, e.g., Wadler 1995: 33). Equivalently, we can (as we do in this paper) use $\rho :: a \to T\,a$, $\circledast :: T\,(a \to b) \to T\,a \to T\,b$, and $\mu :: T\,(T\,a) \to T\,a$, requiring these operations to satisfy the following two laws:

| **Associativity** | **Identity** |
|---|---|
| $\mu \circ \mu = \lambda m.\,\mu\,(\rho\,\mu \circledast m)$ | $\mu \circ \rho = \lambda m.\,\mu\,(\rho\,\rho \circledast m) = \lambda m.\,m$ |

The monad laws for the $\rho$, $\circledast$, and $\mu$ presentation are admittedly complex. The payoff of this presentation, though, is two-fold. First, facilitating composition with $\circledast$ is significantly closer to existing semantic practice than using $\gg\!=$ (recall that we factored $\circledast$, and $\rho$, directly out of the standard account of pronouns).

[5] Writing '$\lambda g.\,\lambda x.\,\mathbf{hates}\,(g_1\,g^{0\to x})$' in lieu of the equivalent '$\lambda g.\,\lambda x.\,\mathbf{hates}\,(g_1^{0\to x}\,g^{0\to x})$'.

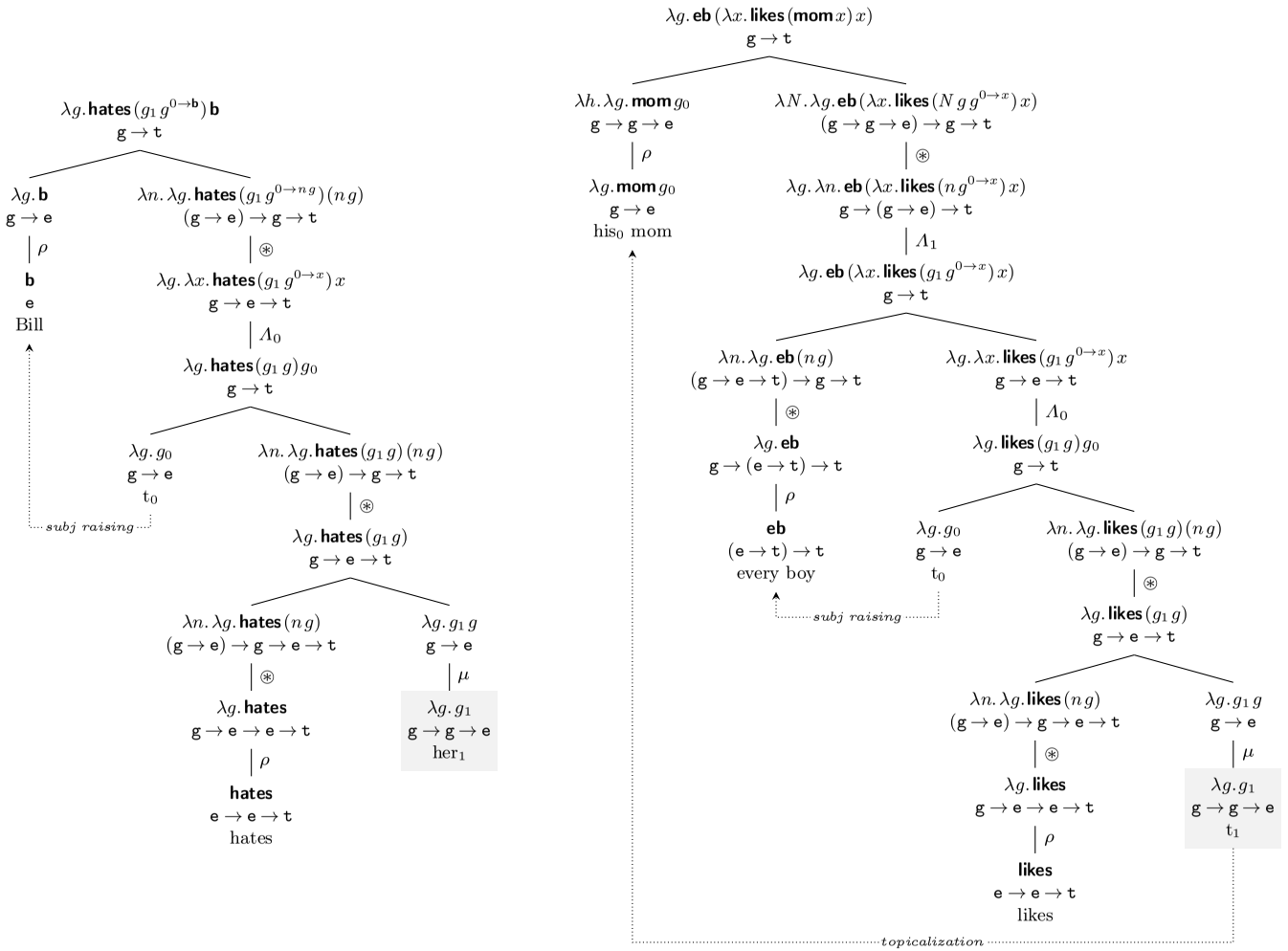


**Fig. 6.** Left: deriving a paycheck reading with a higher-order instantiation of the pronoun and a corresponding application of the monadic $\mu$. Given a $\mathbf{g}$ such that $\mathbf{g}_1 = \lambda g.\,\mathbf{mom}\,g_0$ (i.e., the intension of *his*$_0$ *mom*), the extension of this sentence at $\mathbf{g}$ is $\mathbf{hate}\,(\mathbf{mom}\,\mathbf{b})\,\mathbf{b}$. Right: deriving binding reconstruction with a higher-order trace and a corresponding application of $\mu$. In contrast with the paycheck derivation, here the higher-order variable is resolved to an intra-sentential antecedent, via in-scope binding (precipitated by $\Lambda_1$).

Second, monads generally require *scope-taking* to work their magic: the $a \to T\,b$ argument of $\gg\!=$ must be created by scoping something of type $a$ over something of type $b$, applying $\eta$ to the $b$, and then abstracting over $a$'s 'trace'. While this approach is certainly workable (Charlow 2014, 2017), it's worth exploring the comparatively conservative 'in situ' derivations afforded by $\circledast$ in lieu of $\gg\!=$.

Finally, as the $\rho$, $\circledast$, and $\mu$ formulation of a monad makes clear, every monad determines (at least one) applicative functor. The reverse, however, is not generally true. While (e.g.) $\mathtt{S}$, $\mathtt{G} \circ \mathtt{S}$, and $\mathtt{C}_r$ are all monadic, it turns out that $\mathtt{S} \circ \mathtt{G}$ isn't! (To get a flavor for why, you can try to find a flattener $\mu :: \mathtt{S}(\mathtt{G}(\mathtt{S}(\mathtt{G}a))) \to \mathtt{S}(\mathtt{G}a)$. It's much harder than finding a $\mu :: \mathtt{G}(\mathtt{S}(\mathtt{G}(\mathtt{S}a))) \to \mathtt{G}(\mathtt{S}a)$, and any function you *do* find will run afoul of one of the monad laws above.) In other words, while applicatives are, happily, closed under composition, monads, alas, are not.

## 5 Stepping back, to applicatives

### 5.1 What's the type of an assignment?

We often blithely think of assignments as having type $\mathbb{N} \to \mathtt{e}$. But this is insufficient: (a) it rules out the kinds of assignments we've assumed above, which harbor intensions alongside individuals; and (b) it's incompatible with cross-categorial topicalization, extraposition, scope reconstruction, etc., all of which require us to be able to bind variables with types other than $\mathtt{e}$ (cf. Heim & Kratzer 1998: 213).

Muskens (1995) offers a type-theoretic treatment of assignments as primitive objects that can be used to value variables of arbitrary types. However, Muskens also cautions that special care must be taken when using assignments to value *intensional* variables (as we have done), on pain of inconsistency (cf. Muskens 1995: 179–80). While this technical difficulty has a solution consistent with the outlook of Section 4 (*ibid.*; see also Hardt 1999: 216–7), I think it's illuminating to briefly and tentatively explore a more conservative solution, one which doesn't take assignments as primitives, and which allows intensional variables without any fuss. Moreover, as we will see presently, taking this tack allows us to explain paychecks and reconstruction without monads, in terms of the simpler (and more modular) notion of applicatives.

### 5.2 Higher-order applicatives and higher-order variables

We begin by admitting only type-homogeneous assignments. Formally, we take $\mathtt{g}_r ::= \mathbb{N} \to r$. Thus we have, e.g., assignments for individuals ($\mathtt{g}_\mathtt{e}$), assignments for properties ($\mathtt{g}_{\mathtt{e}\to\mathtt{t}}$), assignments for *intensions* of individuals ($\mathtt{g}_{\mathtt{g}_\mathtt{e}\to\mathtt{e}}$), and so on. Replacing $\mathtt{g}$ *simpliciter* with a hierarchy of assignments means we don't risk packing too much stuff into them and ending up with an inconsistent type theory.

Consider now how we might use $\mathtt{g}_r$ assignments to interpret the bracketed part of (20). We need to value its pronoun, so we'll need a $\mathtt{g}_\mathtt{e}$. We need to value its VP-trace, so we'll need a $\mathtt{g}_{\mathtt{e}\to\mathtt{t}}$. Splitting the difference, we conclude that the bracketed expression should have type $\mathtt{g}_\mathtt{e} \to \mathtt{g}_{\mathtt{e}\to\mathtt{t}} \to \mathtt{t}$.

(20) ... And buy the couch $\Lambda_0$ [she$_1$ did t$_0$].

*Can* we derive something of that type? Yes, by generalizing our previous applicative $\mathtt{G}$ and composing it with itself. We define a type constructor $\mathtt{G}_r\, a ::= \mathtt{g}_r \to a$, and take $\rho$ and $\circledast$ as in (11) and (12), but with generalized types: $\rho :: a \to \mathtt{G}_r\, a$, and $\circledast :: \mathtt{G}_r\,(a \to b) \to \mathtt{G}_r\, a \to \mathtt{G}_r\, b$. Then the applicative functor $\mathtt{G}_\mathtt{e} \circ \mathtt{G}_{\mathtt{e}\to\mathtt{t}}$ (Section 3.3) can be used to derive a doubly assignment-dependent meaning: $\lambda g.\, \lambda h.\, h_0\, g_1$.

The higher-order variables used to model paychecks and reconstruction can be treated analogously to (20). Notice that a higher-order meaning for, e.g., $her_0$ has type $\mathtt{G}_{\mathtt{G}_\mathtt{e}\,\mathtt{e}}\,(\mathtt{G}_\mathtt{e}\,\mathtt{e})$: it depends on an intension-assignment to fix its 'initial' reference, and then on an individual-assignment to fix the value of the retrieved intension.[6] As in (20), we can use a composed applicative (here, $\mathtt{G}_{\mathtt{G}_\mathtt{e}\,\mathtt{e}} \circ \mathtt{G}_\mathtt{e}$) to juggle these two dependencies. The ultimate result for, e.g., the paycheck reading of *Bill hates* $her_0$ will be $\lambda g.\, \lambda h.\, \mathsf{hates}\,(g_1\, h^{0\to\mathtt{b}})\,\mathsf{b}$ — virtually identical to what we monadically derived in Figure 6 (left), but depending on *two* assignments rather than one.

All told, type-homogeneous assignments seem attractive: the underlying type theory is simpler, there's no whiff of inconsistency in the air, and we needn't exploit the extra power of monads to treat paychecks and reconstruction. The only 'price' we pay is needing multiple assignments to extract propositional content from certain utterances.

## 6 A bit of variable-free semantics to play us out

Jacobson (1999) proposes a variable-free account of pronouns that eschews indices and assignments, instead treating pronouns as identity functions: $[\![\text{she}]\!] \coloneqq \lambda x.\, x$.

How should pronominal denotations like these, of type $\mathtt{e} \to \mathtt{e}$, be compositionally integrated, given that they occur in places where something of type $\mathtt{e}$ is expected? The answer may by this point be obvious: after all, the situation is perfectly analogous to the one we found ourselves in before — we've only replaced $\mathtt{g}$-dependent $\mathtt{e}$'s with $\mathtt{e}$-dependent $\mathtt{e}$'s. So we can just use our trusty applicative to compose up a variable-free sentence meaning. Figure 7 gives the details (abstracting away from the elaborations considered in Section 5). In other words, variable-free semantics can be done using the *exact same* combinatory tools as the modular treatment of assignment-sensitivity in variable-full semantics.

Though I lack the space to develop this point as much as it deserves, one other striking fact bears mentioning before we conclude. Consider *she saw her*. A variable-free theory should assign it the following value: $\lambda x.\, \lambda y.\, \mathsf{saw}\, y\, x$. How's that supposed to go, compositionally? Again, a solution presents itself: we can simply compose the variable-free semanticist's applicative functor with itself, exactly as in Section 5! Dependence on two assignments in the variable-full theory becomes dependence on two assignment-free model theoretic objects (here, $\mathtt{e}$'s) in the variable-free theory. Moreover, uncurrying the two-place function derived by the variable-free theory gives $\lambda(x, y).\, \mathsf{saw}\, y\, x = \lambda p.\, \mathsf{saw}\, p_2\, p_1$: assignment-dependence springs organically into being. Thus, not only can variable-free and variable-full approaches to semantics be treated with an identical set of applicative (or, if you

[6] Pronouns and traces will still have a uniform lexical semantics, $[\![\text{pro}_i]\!] \coloneqq \lambda g.\, g_i$, along with a polymorphic type: $\mathtt{pro} ::= \mathtt{G}_\mathtt{e}\,\mathtt{e} \mid \mathtt{G}_{\mathtt{pro}}\,\mathtt{pro}$.

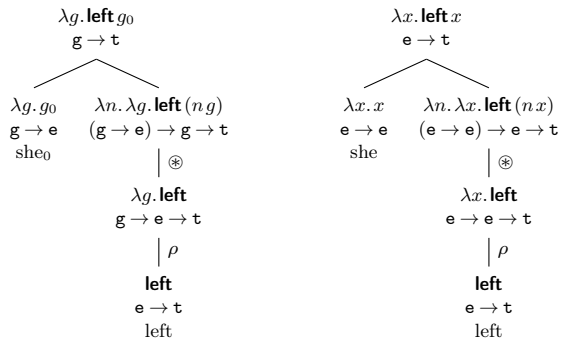


**Fig. 7.** Variable-full (left) and variable-free derivations (right), applicative-style.

prefer, monadic) combinatory tools, modulo types — the semantic values thereby generated turn out to be equivalent up to isomorphism.

## References


Barker, Chris. 2012. Quantificational binding does not require c-command. *Linguistic Inquiry* 43(4). 614–633. https://doi.org/doi:10.1162/ling_a_00108.

Barker, Chris & Chung-chieh Shan. 2014. *Continuations and natural language*. Oxford: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199575015.001.0001.

Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University Ph.D. thesis. http://semanticsarchive.net/Archive/2JmMWRjY/.

Charlow, Simon. 2017. The scope of alternatives: Indefiniteness and islands. Unpublished ms. http://ling.auf.net/lingbuzz/003302.

Cooper, Robin. 1979. The interpretation of pronouns. In Frank Heny & Helmut S. Schnelle (eds.), *Syntax and semantics, volume 10: Selections from the Third Groningen Round Table*, 61–92. New York: Academic Press.

Engdahl, Elisabet. 1986. *Constituent questions*. Vol. 27 (Studies in Linguistics and Philosophy). Dordrecht: Reidel. https://doi.org/10.1007/978-94-009-5323-9.

von Fintel, Kai & Irene Heim. 2011. Intensional semantics, Spring 2011 edition.

Giorgolo, Gianluca & Ash Asudeh. 2012. $(M, \eta, \star)$: Monads for conventional implicatures. In Ana Aguilar Guevara, Anna Chernilovskaya & Rick Nouwen (eds.), *Proceedings of Sinn und Bedeutung 16*, 265–278. MIT Working Papers in Linguistics. http://mitwpl.mit.edu/open/sub16/Giorgolo.pdf.

Groenendijk, Jeroen & Martin Stokhof. 1990. Dynamic Montague grammar. In Laszlo Kalman & Laszlo Polos (eds.), *Proceedings of the Second Symposium on Logic and Language*, 3–48. Budapest: Eötvös Loránd University Press.

Hamblin, C. L. 1973. Questions in Montague English. *Foundations of Language* 10(1). 41–53.

Hardt, Daniel. 1999. Dynamic interpretation of verb phrase ellipsis. *Linguistics and Philosophy* 22(2). 187–221. https://doi.org/10.1023/A:1005427813846.

Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. Oxford: Blackwell.

Jacobson, Pauline. 1999. Towards a variable-free semantics. *Linguistics and Philosophy* 22(2). 117–184. https://doi.org/10.1023/A:1005464228727.

Jacobson, Pauline. 2000. Paycheck pronouns, Bach-Peters sentences, and variable-free semantics. *Natural Language Semantics* 8(2). 77–155. https://doi.org/10.1023/A:1026517717879.

Kennedy, Chris. 2014. Predicates *and* formulas: Evidence from ellipsis. In Luka Crnič & Uli Sauerland (eds.), *The art and craft of semantics: A festschrift for Irene Heim*, vol. 1 (MIT Working Papers in Linguistics), 253–277. http://semanticsarchive.net/Archive/jZiNmM4N/.

Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In Makoto Kanazawa, Lawrence S. Moss & Valeria de Paiva (eds.), *NLCS'15. Third workshop on natural language and computer science*, vol. 32 (EPiC Series), 29–38.

Kobele, Gregory M. 2010. Inverse linking via function composition. *Natural Language Semantics* 18(2). 183–196. https://doi.org/10.1007/s11050-009-9053-7.

Kratzer, Angelika & Junko Shimoyama. 2002. Indeterminate pronouns: The view from Japanese. In Yukio Otsu (ed.), *Proceedings of the Third Tokyo Conference on Psycholinguistics*, 1–25. Tokyo: Hituzi Syobo.

McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. https://doi.org/10.1017/S0956796807006326.

Muskens, Reinhard. 1995. Tense and the logic of change. In Urs Egli, Peter E. Pause, Christoph Schwarze, Arnim von Stechow & Götz Wienold (eds.), *Lexical Knowledge in the Organization of Language*, 147–183. Amsterdam: John Benjamins. https://doi.org/10.1075/cilt.114.08mus.

Poesio, Massimo. 1996. Semantic ambiguity and perceived ambiguity. In Kees van Deemter & Stanley Peters (eds.), *Semantic Ambiguity and Underspecification* (CSLI Lecture Notes 55), 159–201. Stanford: CSLI Publications.

Romero, Maribel & Marc Novel. 2013. Variable binding and sets of alternatives. In Anamaria Fălăuș (ed.), *Alternatives in Semantics*, chap. 7, 174–208. London: Palgrave Macmillan UK. https://doi.org/10.1057/9781137317247_7.

Shan, Chung-chieh. 2002. Monads for natural language semantics. In Kristina Striegnitz (ed.), *Proceedings of the ESSLLI 2001 Student Session*, 285–298. http://arxiv.org/abs/cs/0205026.

Shan, Chung-chieh & Chris Barker. 2006. Explaining crossover and superiority as left-to-right evaluation. *Linguistics and Philosophy* 29(1). 91–134. https://doi.org/10.1007/s10988-005-6580-7.

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.

Unger, Christina. 2012. Dynamic semantics as monadic computation. In Manabu Okumura, Daisuke Bekki & Ken Satoh (eds.), *New Frontiers in Artificial Intelligence JSAI-isAI 2011*, vol. 7258 (Lecture Notes in Artificial Intelligence), 68–81. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-32090-3_7.

Wadler, Philip. 1995. Monads for functional programming. In Johan Jeuring & Erik Meijer (eds.), *Advanced Functional Programming*, vol. 925 (Lecture Notes in Computer Science), 24–52. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-59451-5_2.