Overview: applicative functors are an abstraction that emerged relatively recently in the functional programming literature for dealing with *effectful* computation (Mcbride & Paterson 2008). Given some effectful domain defined by a type constructor F, an applicative provides a way of embedding pure computations into a pure fragment of F's effectful domain, and the peculiar way in which *application* is interpreted within that domain.

Monads are a related, and more established abstraction for dealing with effectful computation (Wadler 1995), and there has been a great deal of work in the linguistics literature motivating an approach to semantic computation using monadic machinery (see, e.g., Shan 2002, Asudeh & Giorgolo 2016 a.o.). Monads are *more powerful* than applicatives – this is because the *bind* operator ($\gg=$) associated with a given monad allows the result of an effectful computation to influence subsequent effects. Applicative functors don't allow this – effects don't influence the structure of computation, they just get sequenced. To quote Mcbride & Paterson (2008: p. 8) "if you need a Monad, that is fine; if you need only an Applicative functor, that is even better!". It is still an open question whether or not natural language semantics requires the full power of *monads*, or if applicative functors suffice.

In the present work, we construct a effectful semantic fragment using the applicative abstraction. Empirically, we focus on the dynamics of anaphora and presupposition projection. We aim to show that dynamics can be modelled with applicatives in a fully modular fashion; we don't need the full power of a monad (c.f. Charlow 2014). We take advantage of the fact that, unlike monads, applicatives *compose* – and the result is guaranteed to be an applicative. Once we introduce the applicative abstraction, it turns out that the machinery necessary for dealing with the dynamics of anaphora and presupposition projection are *already implicit* in the machinery used in an orthodox static setting for dealing with *assignment sensitivity*, *scope*, and *partiality*. In this sense, the present work bears directly on debate surrounding the explanatory power of dynamics for anaphora and presupposition projection (see, e.g., Schlenker 2009).

Many of the ideas here are inspired by de Groote (2006) and Charlow (2014). de Groote (2006) pioneered the approach to dynamic semantics in terms of *continuations*, which will also be a necessary ingredient in the present account. Charlow's (2014) work is foundational for understanding natural language semantics, and specifically dynamics, as *effectful computation*. Unlike the present work, Charlow makes use of monadic machinery – specifically the `State.Set` monad – a technique which provides strictly more expressive power than the applicative abstraction. In the talk, we provide an explicit comparison between the system outlined here and an approach in terms of `State.Set`.

Assignment-sensitivity: Formally, an applicative functor is a tuple, consisting of a type constructor F, a unary operation η of type $\mathsf{a} \to \mathsf{F\,a}$, and a binary operation $\circledast$ of type $\mathsf{F\,(a \to b)} \to \mathsf{F\,a} \to \mathsf{F\,b}$. Applicative functors must obey the following laws:

- Identity: $\mathsf{id}^{\eta} \circledast a = a$
- Homomorphism: $f^{\eta} \circledast x^{\eta} = (f\ x)^{\eta}$
- Composition: $\eta \circ \circledast a \circledast b \circledast c = a \circledast (b \circledast c)$
- Interchange: $a \circledast b^{\eta} = (\lambda f\,.\,f\ b)^{\eta} \circledast a$

Charlow (2018) shows that the applicative abstraction can be marshalled to provide an elegant semantics for pronominal binding that accounts for binding reconstruction and paycheck pronouns, etc. Given G, a type-constructor that specifies the *assignment-sensitive* value-space, we can define

the applicative for assignment sensitivity as in (1);[1] pronouns denote assignment-sensitive individuals (2). An example effectful computation using the applicative operators is given in (3) – $\circledast$ just sequences assignment-sensitivity while retaining the structure of pure computation.

(1) a. $\mathsf{G\,a} \coloneqq \mathsf{g} \to \mathsf{a}$

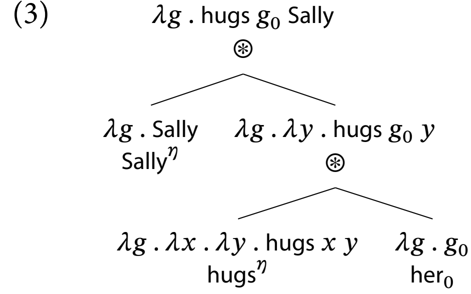
b. $\eta\, a = \lambda g\,.\, a$

c. $n \circledast m = \lambda g\,.\, \mathsf{A}\,(n\,g)\,(m\,g)$

(2) a. $\mathsf{pro}_n \coloneqq \mathsf{G\,e}$

b. $\mathsf{pro}_n = \lambda g\,.\, g_n$

(3)



Composing scope-taking and assignment-sensitivity: Barker & Shan (2014) argue that continuation semantics can provide a general framework for understanding scope in natural language. Although it is ordinarily presented in its monadic guise, the continuation type constructor $\mathsf{K_b}$ also has an applicative instance, which we'll be using to extend our assignment-sensitive fragment to handle dynamics – $\circledast$ sequences *scopal* effects from left-to-right while retaining the structure of pure computation.

(4) a. $\mathsf{K_b\,a} \coloneqq (\mathsf{a} \to \mathsf{b}) \to \mathsf{b}$

b. $\eta\, a = \lambda k\,.\, k\, a$

c. $\mathbb{n} \circledast \mathbb{m} = \lambda k\,.\, \mathbb{n}\,(\lambda n\,.\, \mathbb{m}\,(\lambda m\,.\, k\,(\mathsf{A}\,n\,m)))$

For the most part, Barker & Shan (2014) deal with K parameterized to type t. Here, we'll parameterize K G t, which we can think of as the type of a set of assignments, i.e., a *context*. A continuized individual of type $\mathsf{K_{G\,t}\,e}$ therefore expands to type $(\mathsf{e} \to \mathsf{G\,t}) \to \mathsf{G\,t}$. Unlike monads, applicatives *compose*. Below, we define an applicative functor C – the result of $\mathsf{K_{G\,t}} \circ \mathsf{G}$. The applicative operations associated with C are just the composition of those associated with $\mathsf{K_{G\,t}}$ and G.[2] Inuitively, $\circledast$ sequences scopal effects on the *context* from left-to-right. We can also define a function ↑ for lifting meanings in the assignment-sensitive dimension (G) to an *assignment-sensitive subset* of the C dimension. ↑ therefore behaves like η in a restricted value-space.

(5) a. $\mathsf{C} \coloneqq \mathsf{K_{(G\,t)}} \circ \mathsf{G}$

b. $\eta\, a = \lambda k\,.\, k\,(\lambda g\,.\, a)$

c. $\mathbb{n} \circledast \mathbb{m} = \lambda k\,.\, \mathbb{n}\,(\lambda n[\mathbb{m}\,(\lambda m\, [k\,(\lambda g\,.\, \mathsf{A}\,(n\,g)\,(m\,g))])])$

(6) a. $(\uparrow) \coloneqq \mathsf{G\,a} \to \mathsf{C\,a}$

b. $x^{\uparrow} = \lambda k\,.\, k\,(\lambda\, g\,.\, x\, g)$

The move from a static, assignment-sensitive setting to a dynamic setting boils down to a single *dynamicization operator* ⤊, which converts a function that can manipulate the static context to a function that can manipulate the dynamic context. First, we demonstrate that ⤊ derives *dynamic* ∃

[1] Following Charlow, we defined $\circledast$ in terms of *overloaded* function application A. A applies its first argument to its second, or vice versa, whichever is defined.

[2] $\eta_{\mathsf{K\,t}} \circ \eta_{\mathsf{G}} = \eta_{\mathsf{C}}$. In order to compose two curried binary operators, we just compose the composition operator with itself, i.e., $((\circ) \circ (\circ))\,(\circledast_{\mathsf{K_{G\,t}}})\,(\circledast_{\mathsf{G}}) = (\circledast_{\mathsf{C}})$

from static first-order $\exists$. The intuition here is that the continuation argument k, which dynamic $\exists$ scopes over, represents the *future of the discourse*. Generalizing $\Uparrow$ to n-ary operations allows us to derive dynamic conjunction from η_{G}-shifted static conjunction $\triangle$:

(7) a. $(\Uparrow) \coloneqq (\mathsf{G\,t} \to \mathsf{G\,t}) \to (\mathsf{C\,t} \to \mathsf{C\,t})$

b. $f^{\Uparrow} = \lambda p \,.\, \lambda k \,.\, (f \circ p)\, k$

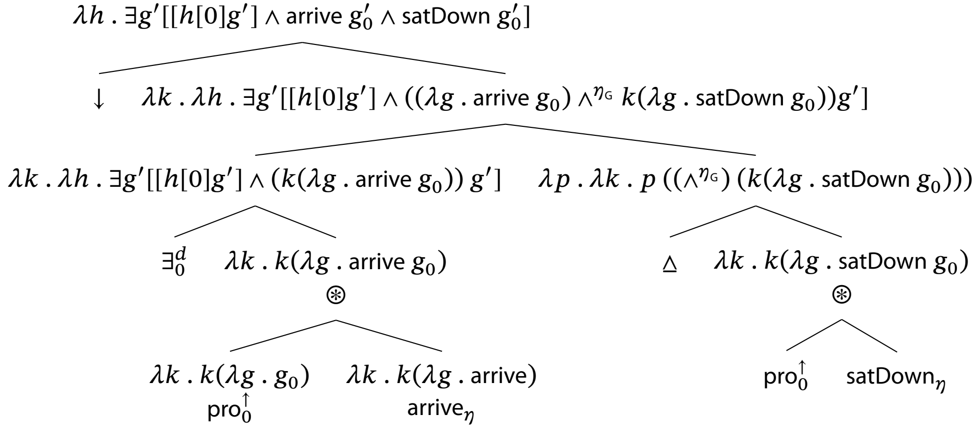
(8) a. $\exists^{s}_{n} = \lambda p \,.\, \lambda g \,.\, \exists g'[g'[n]g \wedge p\, g']$

b. $\exists^{d}_{n} (= \exists^{s,\Uparrow}_{n})$
$= \lambda p \,.\, \lambda k \,.\, \lambda g \,.\, \exists g'[g'[n]g \wedge [p\, k]\, g']$

(9) $(\triangle) = [\lambda q \,.\, \lambda p \,.\, \lambda k \,.\, (p \circ (\wedge^{\eta_{\mathsf{G}}}) \circ q)\, k] \equiv [\lambda q \,.\, \lambda p \,.\, \lambda k \,.\, p\, ((\wedge^{\eta_{\mathsf{G}}})\, q\, k)]$

Now we have everything we need to derive a simple case of donkey anaphora:[3] Note that pronouns just get lifted via $\uparrow$, everything else is handled automatically by the applicative operations provided by C. In the paper, we show how this system can be extended to handle generalized quantification.

(10) Someone^{0} arrived. They$_{0}$ sat down.



Applicatives for presupposition projection: here we follow Shan (2002) in treating *intensionality* as assignment-sensitivity; We define a type-constructor S for world-sensitive meanings. The applicative operations for S are identical to those of G. Predicates are taken to be inherently world-sensitive. World-sensitive computation can be modelled as effectful computation via applicative machinery in *exactly* the same way as assignment-sensitivity.

(11) a. $\mathsf{S\,a} \coloneqq \mathsf{s} \to \mathsf{a}$

b. $\eta\, a = \lambda w \,.\, a$

c. $n \circledast m = \lambda w \,.\, \mathsf{A}\, (n\, w)\, (m\, w)$

(12) a. $\mathsf{arrive} \coloneqq \mathsf{e} \to \mathsf{S\,t}$

b. $\mathsf{arrive} = \lambda x \,.\, \lambda w \,.\, \mathsf{arrive}_w\, x$

In a static setting, presuppositions are typically modelled via partiality (Heim & Kratzer 1998). Unsurprisingly, there's an applicative for that: `Maybe` (here: P), which defines a value-space consisting of defined values $\langle a \rangle$ and an undefined value #. P composes with S to give us the value-space inhabited by presuppositional predicates.

[3]We have one extra shifter here – $\downarrow$ is just the standard way of lowering a continuized meaning by feeding it the identity function: $Q^{\downarrow} = Q\ \mathsf{id}$.

(13) a. $\mathsf{P\ a} \coloneqq \langle \mathsf{a} \rangle \mid \#$
b. $a_\eta = \langle a \rangle$
c. $\langle f \rangle \circledast \langle x \rangle = \langle f\ x \rangle$
else #

(14) a. $\mathsf{stopSmoking} \coloneqq \mathsf{e} \to (\mathsf{S} \circ \mathsf{Maybe})\,\mathsf{t}$
b. $\mathsf{stopSmoking} = \lambda x\,.\,\lambda w\,.\begin{cases}\mathsf{didSmoke}\ x\ w & \langle \mathsf{notSmoke}\ x\ w \rangle \\ \mathsf{else} & \# \end{cases}$

Famously, a static trivalent theory of presuppositions can't provide a satisfactory account of presupposition projection in complex sentences – when the presupposition of the second conjunct is entailed by the first, it fails to project. Here, we demonstrate that we can upgrade our existing fragment to one that accounts for the dynamics of presupposition projection by using the same technique as we used to get dynamics for anaphora - we're going to paramterize our continuation type constructor to *partial propositions*, i.e., $\mathsf{K}_{(\mathsf{S} \circ \mathsf{P})\,\mathsf{t}}$, and compose the result with our type constructor for presuppositional meanings ($\mathsf{S} \circ \mathsf{P}$). Just as before, we can define a lifter $\Uparrow$ to dynamicize meanings. When we apply generalized $\Uparrow$ to $\wedge^{\eta_G}$, we get the Stalnakerian update function $+$.

(15) $\mathsf{U} \coloneqq \mathsf{K}_{(\mathsf{S} \circ \mathsf{P})} \circ (\mathsf{S} \circ \mathsf{P})$

(16) $(+) = \lambda q\,.\,\lambda p\,.\,\lambda k\,.\,(p \circ (\wedge^{\eta_S}) \circ q)\,k$

(17) a. $(\Uparrow) \coloneqq (\mathsf{S\ t} \to \mathsf{S\ t}) \to (\mathsf{U\ t} \to \mathsf{U\ t})$
b. $f^{\Uparrow} = \lambda p\,.\,\lambda k\,.\,(f \circ p)\,k$

Below we demonstrate how this system derives a simple case of local satisfaction. Since the first conjunct entails the presupposition of the second, the complex sentence is effectively presuppositionless.

(18) John used to smoke and he stopped smoking.

$$\downarrow\ \lambda k\,.\,(\lambda w\,.\,\langle \mathsf{didSmoke\ j}\ w \rangle) \wedge^{\eta_S} k\left(\lambda w\,.\begin{cases}\mathsf{didSmoke\ j}\ w & \langle \mathsf{notSmoke\ j}\ w \rangle \\ \mathsf{else} & \# \end{cases}\right)$$
$$\equiv \lambda w\,.\,\langle \mathsf{didSmoke\ j}\ w \wedge \mathsf{notSmoke\ j}\ w \rangle$$

$$\lambda l\,.\,l(\lambda w\,.\,\langle \mathsf{didSmoke\ j}\ w \rangle) \qquad \lambda p\,.\,\lambda k\,.\,p\left((\wedge^{\eta_S})\,k\left(\lambda w\,.\begin{cases}\mathsf{didSmoke\ j}\ w & \langle \mathsf{notSmoke\ j}\ w \rangle \\ \mathsf{else} & \# \end{cases}\right)\right)$$

John did smoke$^{\uparrow}$

$$+ \qquad \lambda k\,.\,k\left(\lambda w\,.\begin{cases}\mathsf{didSmoke\ j}\ w & \langle \mathsf{notSmoke\ j}\ w \rangle \\ \mathsf{else} & \# \end{cases}\right)$$

John stopSmoking

References: ❖ Asudeh, Ash & Gianluca Giorgolo. 2016. Perspectives. *Semantics and Pragmatics* 9. ❖ Barker, Chris & Chung-chieh Shan. 2014. *Continuations and natural language*. Oxford University Press. 228 pp. ❖ Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University dissertation. ❖ Charlow, Simon. 2018. A modular theory of pronouns and binding. unpublished manuscript. ❖ de Groote, Philippe. 2006. Proceedings of SALT 16 16. 1–16. ❖ Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. Malden, MA: Blackwell. 324 pp. ❖ Mcbride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). ❖ Schlenker, Philippe. 2009. Local contexts. *Semantics and Pragmatics* 2. ❖ Shan, Chung-chieh. 2002. Monads for natural language semantics. *arXiv:cs/0205026*. ❖ Wadler, Philip. 1995. Monads for functional programming. In *Advanced functional programming*, 24–52. Springer, Berlin, Heidelberg.