Applicatives for Anaphora and Presupposition

Patrick D. Elliott(✉)

ZAS, Berlin, Germany
patrick.d.elliott@gmail.com

Abstract. In this paper, we construct an effectful semantic fragment using the *applicative* abstraction. Empirically, we focus primarily on the dynamics of anaphora, and secondarily on presupposition projection. We aim to show that a dynamic semantics can be constructed in a fully modular fashion with applicative functors; we don't need the full power of a monad (c.f. [4]). We take advantage of the fact that, unlike monads, applicative functors *compose* – and the result is guaranteed to be an applicative functor. Once we introduce the applicative abstraction, it turns out that the machinery necessary for dealing with the dynamics of anaphora and presupposition projection is *already implicit* in the machinery used in an orthodox static setting for dealing with *assignment sensitivity*, *scope*, and *partiality*.

Keywords: Applicative functors · Dynamic semantics · Presupposition projection · Continuation semantics

1 Overview

Applicative functors are an abstraction for dealing with *effectful* computation that emerged relatively recently in the functional programming literature [10]. 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 [15], and there has already been a great deal of work in the linguistics literature motivating an approach to semantic computation using monadic machinery (see, e.g., [1,4,12]). 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 the choice of subsequent computations. Applicative functors don't allow this – effects don't influence the structure of computation, they just get sequenced. To quote Mcbride and Paterson [10, 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, in order

K. Kojima et al. (Eds.): JSAI-isAI 2018 Workshops, LNAI 11717, pp. 256–269, 2019.
https://doi.org/10.1007/978-3-030-31605-1_19

to model natural language semantics, we require 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, with presupposition projection as a secondary concern. 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. 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., [11]).

Many of the ideas here are inspired by de Groote [6] and Charlow [4]. de Groote [6] 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. We'll offer an explicit comparison between the present approach and Charlow's monadic grammar in Sect. 4.

2 An Applicative for Anaphora

In this section, we begin by introducing some basic building blocks; in Sect. 2.1 we introduce *applicative functors* and the applicative functor laws. In Sect. 2.2, we introduce Charlow's (2018) account of assignment-sensitivity in terms of the applicative instance of `Reader`. In Sect. 2.2, we introduce the applicative instance of `Cont` (more frequently presented in its monadic guise), and following, e.g., Barker and Shan [2], show how it can be marshalled in order to provide a general theory of *scope* in natural language. Finally, in Sect. 2.4 we bring these pieces together in order to account for the dynamics of anaphora. Taking advantage of the fact that, unlike monads, applicatives *compose*, we show how the resources necessary for handling the dynamics of anaphora are *already implicit* in our fragment, once assignment-sensitivity and scope are brought into the picture.

2.1 Applicative Functors

Formally, an applicative functor is a tuple, consisting of a type constructor F, a function `pure` (which we'll write as π) of type $\mathsf{a} \to \mathsf{Fa}$, and a function `apply` (which we'll write as $\circledast$) of type $\mathsf{F(a \to b)} \to \mathsf{Fa} \to \mathsf{Fb}$. Applicative functors must obey the following laws:

(1) Applicative laws

a. Identity: $\mathsf{id}^{\pi} \circledast a = a$

b. Composition: $\pi \circ \circledast a \circledast b \circledast c = a \circledast (b \circledast c)$

c. Homomorphism: $f^{\pi} \circledast x^{\pi} = (f\ x)^{\pi}$

d. Interchange: $a \circledast b^{\pi} = (\lambda f\ .\ f\ b)^{\pi} \circledast a$

Applicative functors are strictly *less powerful* than monads; a monad *is* an applicative functor together with an additional unary operation `join` of type $\mathsf{F}(\mathtt{F}\ \mathsf{a}) \to \mathsf{F}\ \mathsf{a}$, where the applicative's `pure` function corresponds to the monad's `return`, and `apply` corresponds to the monad's `ap` [14,15].[1] The additional power afforded by join means that *monads* can be used to effectively reason about effectful computation, where the results of a previous computation can be used to affect the choice of another. Applicatives, on the other hand, keep the structure of computation fixed, and just sequence effects [10].

Monads have been used to great effect in the linguistic-semantics literature – see, e.g., Shan's [12] pioneering paper, although we note that almost all of the cases discussed by Shan can be recast in terms of applicative functors, and thus the additional power provided by the monadic abstraction isn't strictly speaking motivated. Charlow [4] on the other hand makes crucial use of monadic bind ($\gg\!=$) to account for the exceptional scope of indefinites, and therefore goes some way towards motivating a *monadic* approach to natural language semantics. We'll explicitly compare the fragment outlined here to Charlow's monadic grammar in Sect. 4.

2.2 Assignment Sensitivity

Charlow [3] provides an elegant compositional semantics for pronouns in a static setting via an applicative functor G, defined in (3). G is the type-constructor for the *assignment-sensitive* type space; `g` is the type of assignment functions. It is associated with two functions – π and $\circledast$, defined in (4a) and (4b) respectively.[2,3]. π serves to lift a value to a trivially assignment-sensitive value.

[1] The monad laws are also distinct from the applicative laws, and are generally stated in terms of *monadic bind* ($\gg\!=$), which can be decomposed into ($\circledast$) and `join`. The details are orthogonal to our purposes here.

[2] Note that (4b) is defined in terms of *overloaded function application* A.

(2) a. $\mathsf{A}fx := fx$ $\quad (\mathsf{a} \to \mathsf{b}) \to \mathsf{a} \to \mathsf{b}$

b. $\mathsf{A}xf := fx$ $\quad \mathsf{a} \to (\mathsf{a} \to \mathsf{b}) \to \mathsf{b}$

[3] At various points, it will be important to disambiguate between, e.g., the `pure` functions associated with two different applicative functors. In this case we use a subscript, e.g., the `pure` function associated with G can be written π_G.

$\circledast$ specifies how *application* is interpreted within the assignment-sensitive domain. Pronouns are interpreted as inherently assignment-sensitive individuals, as defined in (5). Figure 1 provides a sample derivation, for the sentence *Sally hugs her*, illustrating how π and $\circledast$ facilitate assignment-sensitive composition.

(3) $\mathsf{Ga} ::= \mathsf{g} \rightarrow \mathsf{a}$

(4) a. $a^{\pi} := \lambda g\,.\,a$ $\qquad \mathsf{a} \rightarrow \mathsf{G\,a}$

b. $n \circledast m := \lambda g\,.\,\mathsf{A}\,(n\ g)(m\ g)$ $\qquad \left.\begin{array}{l}\mathsf{G(a \rightarrow b) \rightarrow G\,a}\\ \mathsf{G\ a \rightarrow G(a \rightarrow b)}\end{array}\right\} \rightarrow \mathsf{G\ b}$

(5) $\mathsf{pro}_n = \lambda g\,.\,g_n$ $\qquad \mathsf{G\ e}$

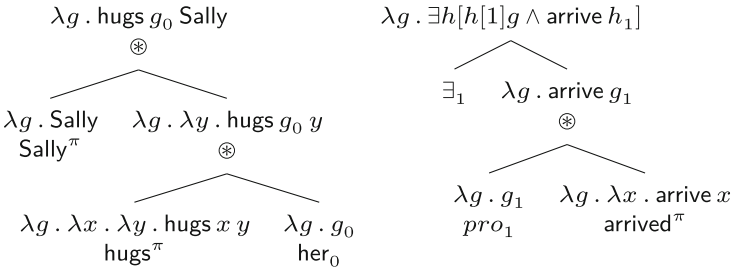

Fig. 1. Assignment-sensitive composition via π and $\circledast$.

It will be useful for subsequent sections to illustrate how we can define first-order quantification in terms of the machinery outlined here. First-order existential quantification is defined standardly as in (6), and first-order universal quantification in (7). Note that $h[n]g$ means that h differs *at most* from g in the value h assigns to n, which we write h_n. See Fig. 1 for a sample derivation of an existential statement such as *someone arrived*. For ease of exposition, at this stage we assume that first-order quantifiers bind silent pronominal traces.

(6) $\exists_n = \lambda p\,.\,\lambda g\,.\,\exists h[h[n]g \wedge ph]$ $\qquad \mathsf{Gt} \rightarrow \mathsf{Gt}$

(7) $\forall_n = \lambda p\,.\,\lambda g\,.\,\forall g'[g'[n]g \rightarrow pg']$ $\qquad \mathsf{Gt} \rightarrow \mathsf{Gt}$

2.3 Continuations and Scope

In this section, we will provide a basic overview of *continuation semantics* through the lens of the applicative functor $\mathsf{K_b}$, defined in (8).[4] Note that, unlike our previous type-constructor G, K comes with an additional type parameter b. The definitions of the `pure` and `apply` operators associated with K are given in (9a) and (9b) respectively. `pure` lifts a value a to a trivially scope-taking value – in fact it is essentially a polymorphic formulation of *Montague Lift*. Again, `apply` specifies how function application is interpreted within the scopal domain.

(8) $\mathsf{K_b\,a} ::= (\mathsf{a} \to \mathsf{b}) \to \mathsf{b}$

(9) a. $a^{\pi} := \lambda k\,.\,ka$ $\qquad \mathsf{a} \to \mathsf{K\,a}$

b. $\mathbb{n} \circledast \mathbb{m} := (\lambda k\,.\,\mathbb{n}(\lambda n\,.\,\mathbb{m}(\lambda m\,.\,k(\mathsf{A}\,n\,m))))$ $\qquad \left.\begin{array}{l}\mathsf{K(a \to b) \to K\,a}\\ \mathsf{K\,a \to K(a \to b)}\end{array}\right\} \to \mathsf{K\,b}$

We can equivalently write the functions associated with $\mathsf{K_b}$ using Barker and Shan's *tower notation*, as in Fig. 2. We'll often take advantage of the relative succinctness of the tower notation, although bear in mind that a tower can always be expanded to a representation in the lambda calculus (see [2] for details). `pure` takes a value and returns a trivial tower; `apply` takes two towers, sequences scopal side-effects from left-to-right, and applies the inner values.

$$a^{\pi} := \frac{[\,]}{a} \qquad \frac{\mathbb{n}\,[\,]}{n} \circledast \frac{\mathbb{m}\,[\,]}{m} := \frac{\mathbb{n}\,[\,\mathbb{m}\,[\,]]}{\mathsf{A}\,n\,m}$$

Fig. 2. The operations associated with the continuation applicative in tower notation

Within continuation semantics, quantificational DPs such as *everyone* are *continuized individuals* of type $\mathsf{K_t\,e}$, as in (10). In order to get back from the scopal tier to an ordinary value, we'll need one final piece of machinery: a lowering function $\downarrow$, which applies a trivially continuized value of type t (since here, $\mathsf{b} = \mathsf{t}$) to the identity function id, as in (11). Figure 3 illustrates composition of a scopal value via $\mathsf{K_t}$.[5]

(10) $\mathsf{everyone} := \frac{\forall x[\,]}{x}$ $\qquad \mathsf{K_t\,e}$

(11) $\downarrow \mathbb{t} := \mathbb{t}\,\mathsf{id}$ $\qquad \mathsf{K_b\,b \to b}$

[4] Out of necessity, our presentation of continuation semantics will presuppose a certain degree of familiarity with the framework, but see Barker and Shan [2] for a thorough introduction.

[5] It is easy to see that if we compose more than one scopal value, the resulting value will correspond to the *surface scope* reading of a given sentence. In order to derive inverse scope readings we need an additional operation – *internal lift*. This won't be relevant for our purposes, but see [2] for details.

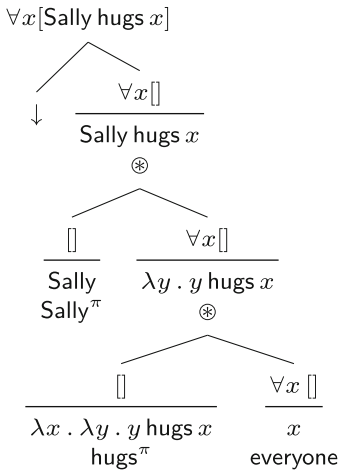

Fig. 3. Composition with a quantificational DP via $\mathsf{K_t}$

2.4 The Dynamics of Anaphora

In the previous section, we parameterized our continuation type-constructor K to t, the type of *truth values*. We did this in order to account for the composition of quantificational DPs such as *everyone*. Let's shift perspective, and instead parameterize K to $\mathsf{G\,t}$, the type of *assignment-sensitive truth values*. Due to the equivalence between characteristic functions of type $\mathsf{a} \to \mathsf{t}$ and sets of type $\{\,\mathsf{a}\,\}$, we can also think of $\mathsf{G\,t}$ as the type of a *set of assignment functions*. In the following, it will be helpful to switch back and forth between characteristic function and set perspectives, although bear in mind that the underlying compositional apparatus uses functions exclusively. The `pure` operator associated with $\mathsf{K_{G\,t}}$ lifts a value a to a scopal value of type $(\mathsf{a} \to \mathsf{G\,t}) \to \mathsf{G\,t}$ (Fig. 4).

Recall that one property of applicative functors is that they compose, and the result is guaranteed to be an applicative functor. The next step in the analysis is to compose $\mathsf{K_{G\,t}}$ and our type-constructor for assignment-sensitivity G, yielding a new applicative functor, which we'll call C, defined in (12). C essentially *is* our analysis of the dynamics of anaphora, so it's worth paying attention at this point. The `pure` and `apply` operators associated with C are just the *composition*

$$a^{\pi} := \frac{[]}{\lambda g\,.\,a} \qquad \frac{\mathbb{m}\,[]}{m} \circledast \frac{\mathbb{n}\,[]}{n} = \frac{\mathbb{m}\,[\,\mathbb{n}\,[]]}{\lambda g\,.\,\mathsf{A}\,(m\;g)\,(n\;g)}$$

Fig. 4. The π and $\circledast$ operators associated with C.

of those associated with $\mathsf{K_{G\,t}}$ and G.[6] We've provided the partially de-sugared definition in (13) for ease of exposition.

(12) $\mathsf{C} ::= \mathsf{K_{G\,t}} \circ \mathsf{G}$

(13) $\mathsf{C\,a} := (\mathsf{G\,a} \to \mathsf{G\,t}) \to \mathsf{G\,t}$

We'll be using C as our type-constructor for the domain of *contextually dynamic values*. In order to get started, let's define a function $(\uparrow)$ that lifts assignment-sensitive values to trivially dynamic values.

(14) $x^{\uparrow} := \frac{[]}{\lambda g\,.\,x\,g}$ $\qquad \mathsf{G\,a} \to \mathsf{C\,a}$

We'll use $\uparrow$ to lift pronouns into the contextually dynamic space, as illustrated below. It turns out that in our new dynamic setting, the fundamental semantic contribution of a pronominal is no different.

(15) $\mathsf{pro}_n^{\uparrow} := \frac{[]}{\lambda g\,.\,g_n}$

As a first attempt at a genuinely dynamic semantics for anaphora, we'll define a function $\Uparrow$, the role of which is to *dynamize* first-order operators such as $\exists$. $\Uparrow$-lifting first-order existential quantification yields *dynamic* existential quantification.

(16) $f^{\Uparrow} := \lambda p\,.\,\lambda k\,.\,(f \circ p)k$ $\qquad (\mathsf{G\,t} \to \mathsf{G\,t}) \to (\mathsf{C\,t} \to \mathsf{C\,t})$

(17) $\exists_n^d := \lambda p\,.\,\lambda k\,.\,\lambda g\,.\,\exists h[h[n]g \wedge (p\,k)h]$ $\qquad \mathsf{C\,t} \to \mathsf{C\,t}$

Once we generalize $\Uparrow$ to binary operations (details suppressed), and apply it to type-lifted static conjunction, we can derive *dynamic conjunction*. The definition of dynamic conjunction will probably look quite unfamiliar to dynamic semanticists used to theories such as Dynamic Predicate Logic [8] and Predicate Logic with Anaphora [7], but it bears some similarities to the definition given in Chierchia [5].

(18) $(\wedge^d) = \lambda q\,.\,\lambda p\,.\,\lambda k\,.\,(p \circ (\wedge_{\mathsf{G}}) \circ q)k$ $\qquad \mathsf{C\,t} \to \mathsf{C\,t} \to \mathsf{C\,t}$

For completeness, we define two more useful operators: *dynamic negation*, and *discourse referent introduction* (dref-intro), as in (19) and (20) respectively. Dynamic negation is defined in a reasonably standard way – the closure operator $\downarrow$ closes off the anaphoric potential of its prejacent. The dref-intro operator shifts an individual denoting expression into a dynamic binder.

[6] For the unary operation associated with each applicative functor this is straightforward: $\pi_{\mathsf{K_{G\,t}}} \circ \pi_{\mathsf{G}} = \pi_{\mathsf{C}}$. In order to compose two curried binary operators however, we compose the composition operator with itself, i.e., $((\circ) \circ (\circ))(\circledast_{\mathsf{K_{G\,t}}})(\circledast_{\mathsf{G}}) = (\circledast_{\mathsf{C}})$.

(19) $(\neg^d) = \lambda p.\uparrow(\neg(\downarrow\ p))$ $\quad \mathsf{C}\,\mathsf{t} \to \mathsf{C}\,\mathsf{t}$

(20) $\Delta_n = \lambda x\,.\,\lambda p\,.\,\lambda k\,.\,\lambda g\,.\,(g^{[n\to x]} \circ p)\,k$ $\quad \Delta_n := \mathsf{e} \to \mathsf{C}\,\mathsf{t} \to \mathsf{C}\,\mathsf{t}$

We're now in a position to derive a simple case of cross-sentential anaphora: *someone arrived and they sat down*. The full computation is illustrated in Fig. 5. Intuitively, the continuation variable k represents the *future of the discourse*. The way that dynamic conjunction passes the continuation variable from one conjunct to the next means that the scope of dynamic existential quantification extends from left-to-right automatically – the continuation variable always ends up taking scope over the right-most conjunct. In the sample derivation, the resulting continuized value is closed off via $(\downarrow)$, returning a familiar assignment-sensitive value.

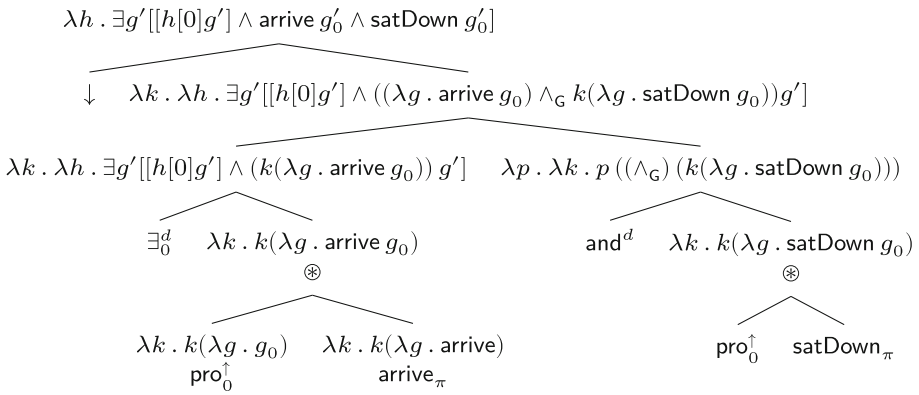

Fig. 5. Example of simple cross-sentential dynamic binding via C

One nice feature of this system is that static existential quantification and conjunction can be dynamicized in a regular way. Inherently assignment-sensitive expressions such as pronominals can be lifted into the contextually dynamic space via a simple lifter function, and every other aspect of composition is handled by the `pure` and `apply` operations associated with C. As such, this framework addresses some of the worries raised about dynamic approaches to anaphora and other phenomena, namely that they must stipulate the dynamic flow of information as part of the lexical entry of each individual expression (see, e.g., [11]).

For completeness, in Fig. 6 we show how dynamic negation roofs the scope of a dynamic existential quantifier by closing off the continuation variable before re-opening it. This directly captures the fact *nobody arrived and they sat down* is unacceptable under the intended reading.

Ultimately, we would like to extend this framework to a full compositional semantics for determiners and generalized quantification. We will leave this

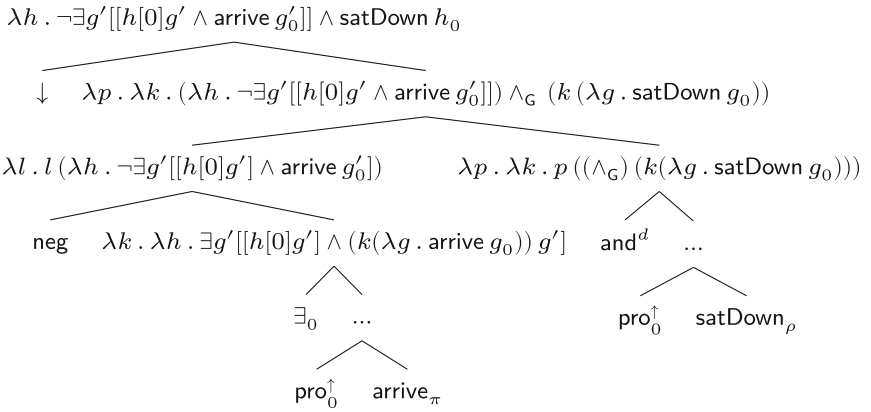

Fig. 6. Dynamic negation blocks dynamic binding

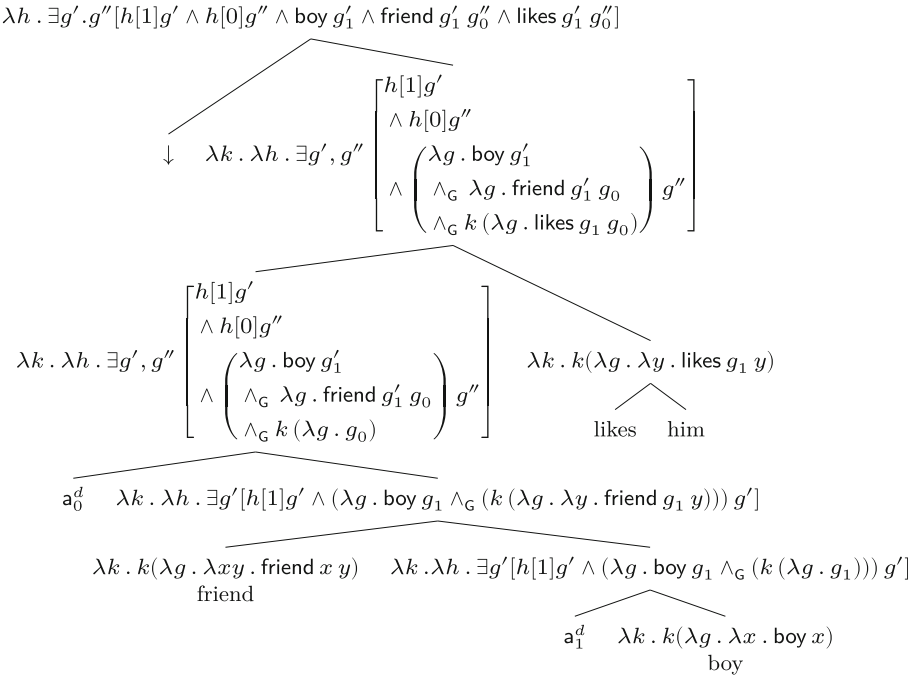

Fig. 7. Dynamic binding from out of a nominal restrictor

mostly for future work, but we do offer a sketch of an analysis of a case involving dynamic binding from out of a nominal restrictor, which relies on the dynamic denotation for the indefinite article given in (21). The computation for the sentence *a friend of a boy likes him*, where *a boy* binds *him*, is given in Fig. 7.

(21) $\mathsf{a}_n^d = \lambda p\,.\,\lambda k\,.\,\exists_n^s((p\,\mathsf{pro}_n^{\uparrow})((\wedge_{\mathsf{G}})(k\,\mathsf{pro}_n)))$ $\qquad \mathsf{C}(\mathsf{e} \to \mathsf{t}) \to \mathsf{C}\,\mathsf{e}$

3 An Applicative for Presupposition Projection

In this section, we argue that the same technique we used to model the dynamics of anaphora can be used to model the dynamics of *presupposition projection*. This will be even more of a proof of concept than the previous section.

3.1 Back to Assignment Sensitivity

The first component we will need in order to get our analysis off the ground is a way of modelling *world-sensitivity/intensionality*. Here we follow Shan [12] 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. If we assume that predicates take an inner world argument, we don't even particularly need π and $\circledast$ to aid in composition.

(22) a. $\mathsf{Sa} ::= \mathsf{s} \to \mathsf{a}$

b. $a^{\pi} := \lambda w\,.\,a$

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

(23) a. $\mathsf{arrive} ::= \mathsf{e} \to \mathsf{S}\,\mathsf{t}$

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

3.2 Partiality

In a static setting, presuppositions are typically modelled via partiality/trivalence [9]. Unsurprisingly, there's an applicative for that: `Maybe` (here: P), which defines a trivalent value-space consisting of defined values $\langle a \rangle$ and an undefined value $\#$.

(24) a. $\mathsf{P\,a} ::= \langle \mathsf{a} \rangle \mid \#$

b. $a^{\pi} := \langle a \rangle$

c. $m \circledast n :- \begin{cases} \langle \mathsf{A}\,x\,y \rangle & \langle x \rangle := m; \langle y \rangle := n \\ \# & \text{otherwise} \end{cases}$

Once we compose S with P ($\mathsf{S}_{\#}$), we end up with the resources we need to model presuppositional, world-sensitive predicates in a static way, as illustrated by the lexical entry for the presuppositional predicate *stop smoking*, given in (26), which is modelled as a function from individuals to partial propositions.

(25) a. $\mathsf{S}_{\#} := \mathsf{S} \circ \mathsf{P}$

b. $a^{\rho} := \lambda w \,.\, \langle a \rangle$

c. $f \circledast x := \lambda w. \begin{cases} \langle gy \rangle & f\, w = \langle g \rangle \wedge x\, w = \langle y \rangle \\ \mathsf{else} & \# \end{cases}$

(26) a. $\mathsf{stopSmoking} := \mathsf{e} \to \mathsf{S}_{\#}\, \mathsf{t}$

b. $\mathsf{stopSmoking} = \lambda x \,.\, \lambda w \,.\, \begin{cases} \mathsf{didSmoke}\, x\, w & \langle \mathsf{notSmoke}\, x\, w \rangle \\ \mathsf{else} & \# \end{cases}$

3.3 The Dynamics of Presupposition Projection

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 [13].

(27) a. If Sally used to smoke, then she stopped smoking.

b. Sally used to smoke, and she stopped smoking.

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 parameterize our continuation type constructor to *partial propositions*, i.e., $\mathsf{K}_{\mathsf{S}_{\#}\,\mathsf{t}}$, and compose the result with our type constructor for presuppositional meanings $\mathsf{S}_{\#}$, giving us the type constructor U, defined in (28). Again, a partially de-sugared definition is given in (29).

Just as before, we can define a lifter $\Uparrow$ to dynamicize meanings. When we apply generalized $\Uparrow$ to $\wedge^{\pi_{\mathsf{G}}}$, we get the Stalnakerian update function $+$.

(28) $\mathsf{U} := \mathsf{K}_{\mathsf{S}_{\#}\,\mathsf{t}} \circ \mathsf{S}_{\#}$

(29) $\mathsf{U\,a} := (\mathsf{S}_{\#}\,\mathsf{a} \to \mathsf{S}_{\#}\,\mathsf{t}) \to \mathsf{S}_{\#}\,\mathsf{t}$

(30) a. $(\Uparrow) := (\mathsf{S\,t} \to \mathsf{S\,t}) \to (\mathsf{U\,t} \to \mathsf{U\,t})$

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

(31) $(+) = \lambda q\,.\,\lambda p\,.\,\lambda k\,.\,(p \circ (\wedge) \circ q)k$

In Fig. 8 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.

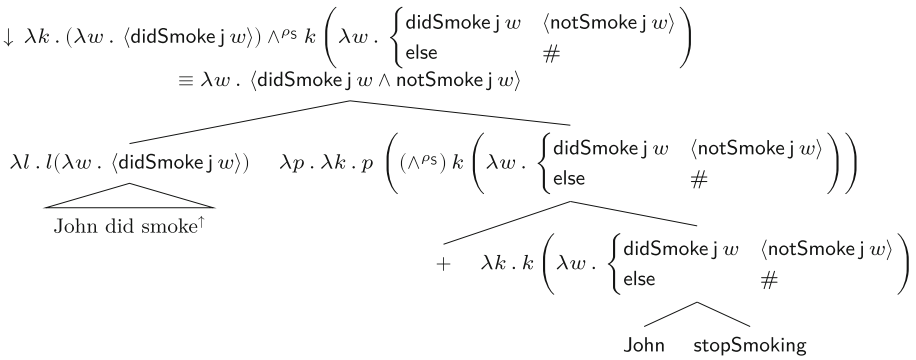

Fig. 8. Local satisfaction via U

4 Comparison to Charlow (2014)

Charlow's monadic grammar has a far broader empirical remit than out fairly modest goal – to decompose dynamics into scope-taking and assignment-sensitivity – allowed for. Therefore, it is difficult to compare the two directly. Nevertheless, we will attempt here to give a flavour of Charlow's approach, and point out some respects in which it differs from the applicative grammar outlined here.

One of Charlow's main goals is to provide a semantics for *indefinites* which accounts both (a) for their ability to take *exceptional scope*, and (b) their dynamic properties. At the core of Charlow's account is the `State.Set` monad, defined in (32) in terms of its `return` (32b) and `bind` (32c) functions. `State.Set` combines the `State` monad and the `Set` monad via the `StateT` monad transformer, in order to capture state-sensitivity (i.e., dynamics), and nondeterminism (i.e., indefiniteness) respectively. Indefinites are given a different semantic treatment to truly quantificational DPs – they treated as individuals with nondeterministic side-effects, as in (33). Quantificational DPs, on the other kind, must be assigned inherently scopal denotations, as in (34).

(32) a. $\mathsf{SS}\, a ::= g \to \{\, \langle a, g \rangle \,\}$

b. $a^{\rho} := \lambda g \,.\, \{\, \langle a, g \rangle \,\}$

c. $m \gg\!= k := \lambda g \,.\, \bigcup_{\langle a, s' \rangle \in m\, s} k\, a\, s'$

(33) $\mathsf{someone} = \lambda g \,.\, \{\, x, \widehat{gx} \mid \mathsf{person}\, x \,\}$ $\quad \mathsf{SS}\, \mathsf{e}$

(34) $\mathsf{everyone} = \frac{\mathsf{ev}\, (\lambda x[])}{x}$ $\quad \frac{\mathsf{SS}\, \mathsf{t}}{\mathsf{e}}$

Charlow [4] argues for a theory of *scope islands* inspired by the concept of *delimited control* in the computer science literature (see, e.g., [14]). The idea, in a nutshell, is that a scope-island is a constituent that must be completely evaluated – in other words, every continuation argument *must* be saturated. This captures the sensitivity of inherently scopal expressions, such as universals, to scope islands – the scopal side effects associated with (34) must be evaluated inside of a given scope island. Indefinites, on the other hand, trigger non-deterministic side-effects that may survive evaluation, thus capturing the ability of indefinites to take apparently exceptional scope. We suppress the details of the analysis here out of necessity.

The applicative grammar outlined here fails to make a distinction between indefinites and quantificational DPs in this respect. In fact, there is nothing to stop us from applying our *dynamicization* operator to a universal quantifier in order to yield a "dynamic" universal, as below. This must be blocked as a lexical stipulation. The exceptional status of indefinites can be considered an argument in favour of the monadic approach of Charlow [4].

(35) $\forall_n^d := \lambda p \,.\, \lambda k \,.\, \lambda g \,.\, \forall h[h[n]g \to (p\, k)h]$

5 Conclusion

In this paper, we've attempted to provide a dynamic theory of anaphora and presupposition projection that is *fully modular* in nature – in fact, the expressivity we needed to capture these phenomena was *already* implicit in our most basic applicative functors for dealing with assignment-sensitivity, scope, world-sensitivity, and partiality. We leave an elaboration of this framework to future work.

Acknowledgments. Thanks to audiences at LENLS15 and an internal ZAS workshop for their attentiveness and feedback, as well as to Simon Charlow for much useful discussion.

References

1. Asudeh, A., Giorgolo, G.: Perspectives. Semant. Pragmat. **9**, 1–57 (2016)
2. Barker, C., Shan, C.-c.: Continuations and Natural Language. Oxford University Press, Oxford (2014)
3. Charlow, S.: A modular theory of pronouns and binding
4. Charlow, S.: On the semantics of exceptional scope (2014)
5. Chierchia, G.: Dynamics of Meaning. University of Chicago Press, Chicago (1995)
6. de Groote, P.: Proceedings of SALT 16. Linguistic Society of America, pp. 1–16. Cornell University, Ithaca, NY (2006)
7. Dekker, P.: Predicate logic with anaphora. Semant. Linguist. Theory **4**, 79–95 (1994)
8. Groenendijk, J., Stokhof, M.: Dynamic predicate logic. Linguist. Philos. **14**(1), 39–100 (1991)
9. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell, Malden (1998)
10. Mcbride, C., Paterson, R.: Applicative programming with effects. J. Funct. Program. **18**(1), 1–13 (2008)
11. Schlenker, P.: Local contexts. Semant. Pragm. **2**, 1–78 (2009)
12. Shan, C.-c.: Monads for natural language semantics. arXiv:cs/0205026 (2002)
13. Stalnaker, R.: Propositions. In: MacKay, A.F., Merrill, D.D. (eds.) Issues in the Philosophy of Language, pp. 79–91. Yale University Press, New Haven (1976)
14. Wadler, P.: Monads and composable continuations. LISP Symb. Comput. **7**(1), 39–55 (1994)
15. Wadler, P.: Monads for functional programming. In: Jeuring, J., Meijer, E. (eds.) AFP 1995. LNCS, vol. 925, pp. 24–52. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59451-5_2