Algebraic effects for extensible dynamic semantics

Julian Grove and Jean-Philippe Bernardy
Centre for Linguistic Theory and Studies in Probability
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
Gothenburg, Sweden

Published in the *Journal of Logic, Language and Information*:
http://dx.doi.org/10.1007/s10849-022-09378-7

Abstract Research in dynamic semantics has made strides by studying various aspects of discourse in terms of computational effect systems, for example, monads (Shan, 2002; Unger, 2011; Charlow, 2014), continuations (de Groote, 2001; Barker and Shan, 2014), and general effect handlers (Maršik, 2016). We provide a system, based on graded monads, that synthesizes insights from these programs by formalizing individual discourse phenomena in terms of separate effects, or grades. Included are effects for introducing and retrieving discourse referents, non-determinism for indefiniteness, and generalized quantifier meanings. We formalize the behavior of individual effects, as well as the interactions between effects, in terms of algebraic laws tailored to the relevant discourse phenomena. The system we propose is thus modular and suggests a novel approach to integrating formal accounts of distinct semantic phenomena. Finally, we give an interpretation of the system into pure λ-calculus that respects the laws. Future work will aim to integrate more discourse phenomena using the same methodology, for example, presupposition and conventional implicature.

1 Introduction

In the last two decades, research in dynamic semantics has attained a breadth of insights into the relation between compositional semantics and dynamic semantics by viewing *prima facie* non-compositional phenomena as arising from computational side effects. The effectful approach to meaning allows one to view compositionally recalcitrant features — e.g., discourse referents, intensionality, and scope-taking — as giving rise to a rich, but uniform structure which integrates them with truth-conditional

meaning. This integration is done by *injecting* truth-conditional meaning into the effectful structure in a natural way, such that the structure gives rise to a *functor*. This pattern of analysis has come in a variety of forms: *continuations* to study scope (de Groote, 2001; Barker, 2002; Barker and Shan, 2014), *graded applicative functors* to study quantification (Kobele, 2018a), and *monads* to study discourse referents, anaphora, and indefiniteness, among other phenomena (Shan, 2002; Giorgolo and Unger, 2009; Giorgolo and Asudeh, 2012; Charlow, 2014, 2020a,b, i.a.).

Progress in understanding dynamic and scopal phenomena in terms of effects, however, has presented two basic methodological questions. On the one hand, given effectful treatments of individual phenomena (say, discourse referents, quantification, and conventional implicature), how does one integrate them into a semantic analysis encompassing them all? On the other hand, how does one study *interactions* between these phenomena, while simultaneously preserving their individual treatments in the result? In this paper, we address both questions by providing a general framework, based on *algebraic effects*, for characterizing individual dynamic semantic phenomena, as well as their interactions, in terms of algebraic laws.

Stated in other terms, our goal is to improve the compositionality properties of functor-based theories of dynamic semantics; i.e., by recasting them as algebraic theories:

- At a meta-theoretical level, when two phenomena are described by two distinct theories within our framework, we provide a systematic recipe for obtaining a combined theory of both phenomena. The combination is monotonic, in the sense that the predictions of the original theories, regarding either phenomenon, remain unchanged in the combined theory.

- In individual analyses, when two syntactically adjacent constituents feature two distinct (yet possibly interacting) phenomena, their meanings may always be combined compositionally, in order to obtain a meaning for their combination.

We begin in §2 with a background to monadic dynamic semantics, and we present the issue of compositionality that pertains to monads and monad transformers. In §3, we axiomatize our approach in terms of the meta-language we use to describe algebraic theories, and we show how meanings may be provided in terms of this meta-language. §4 provides an interpretation of this axiomatizations in terms of a simply typed λ-calculus with products. We discuss related work in §5 before concluding in §6.

2 Monadic dynamic semantics

Since the work of Shan (2002), monads have provided a popular interface for semantic analyses employing computational effects. Monads have been used to study anaphora (Giorgolo and Unger, 2009) and conventional implicature (Giorgolo and Asudeh, 2012), and have more recently been taken up by Charlow (2014, 2020a,b) to study the interactions among quantification, anaphora, indefiniteness, and binding in a framework that relies on monad transformers.

We assume a general familiarity with monads, but we briefly remind the reader of their structure, in order to introduce notation. A monad M is an endofunctor that takes a given type α onto a type $M\alpha$ of computations exhibiting structure that encapsulates some desired side effect, e.g., reading and writing to a store, or non-determinism.[1] Each monad M is associated with two operators, η ('return') and $\star$ ('bind'), having the following type signatures, for any types α and β:

$$\eta : \alpha \to M\alpha$$
$$(\star) : M\alpha \to (\alpha \to M\beta) \to M\beta$$

The role of η is to inject *pure* (i.e., non-effectful) values into the structure provided by M, while $\star$ sequences a computation of type $M\alpha$ with an indexed computation of type $\alpha \to M\beta$ to produce a sequenced computation of type $M\beta$.

2.1 Using monad transformers: Charlow (2014)

Charlow (2014) introduces a monadic dynamic semantics that combines analyses of anaphora, indefiniteness, and quantification by relying on monad transformers. In particular, Charlow uses a Powerset monad to characterize indefiniteness, and then applies a State monad transformer, in order to obtain a system to characterize both indefiniteness and anaphora in the same grammar. He then applies a Continuation monad transformer, in order to provide a setting to study quantification. Crucially, the analyses that he provides for individual phenomena are extended compositionally to obtain analyses of their combinations with new phenomena.[2]

The Powerset monad P allows one to analyze indefinite noun phrases (and the expressions with which they compose) as denoting sets, encoded as functions of type

[1] The underlying category we employ will invariably be Cartesian closed. One may restrict attention to the category of sets and functions, for example.

[2] We use slightly different terminology and notation than that found in Charlow (2014).

$\alpha \to t$:

$$\mathsf{P}\alpha = \alpha \to t$$

$$\eta : \alpha \to \alpha \to t$$
$$\eta a = \{a\} \qquad (= \lambda x.x = a)$$

$$(\star) : (\alpha \to t) \to (\alpha \to \beta \to t) \to \beta \to t$$
$$m \star k = \bigcup_{x \in m} kx \qquad (= \lambda y.\exists x : mx \wedge kxy)$$

This way, the noun phrase *a linguist*, for instance, will denote the set $\{x \mid \mathsf{ling}x\}$ and may be composed with an intransitive verb such as *sleeps* by injecting the latter into the monad via η: $\eta\mathsf{sleep}$. To compose them, Charlow employs monadic functional application (which he overloads with forward and backward application, to be disambiguated by the types of arguments). Functional application (**FA**) is defined as follows for an arbitrary monad M:

$$\begin{aligned} \mathbf{FA} \;:\; & M(\alpha \to \beta) \to M\alpha \to M\beta \\ \text{or}\;\; & M\alpha \to M(\alpha \to \beta) \to M\beta \\ \mathbf{FA}\, m\, n \;=\; & m \star \lambda f.n \star \lambda x.\eta(fx) \\ \text{or}\;\; & m \star \lambda x.n \star \lambda f.\eta(fx) \end{aligned}$$

Now, *a linguist sleeps* may be interpreted as $\mathbf{FA}\{x \mid \mathsf{ling}x\}(\eta\mathsf{sleep})$, which can be reduced to $\{\mathsf{sleep}x \mid \mathsf{ling}x\}$; that is, a set of truth values containing *True* iff some linguist sleeps.

To incorporate anaphora, he invokes the following State monad transformer, which takes an underlying monad M onto a new monad $\mathsf{S}_T M$, for some fixed type s of *states*:

$$\mathsf{S}_T\, M\, \alpha = s \to M(\alpha \times s)$$

$$\eta : \alpha \to s \to M(\alpha \times s)$$
$$\eta a = \lambda s.\eta\langle a, s\rangle$$

$$\begin{aligned} (\star) \;:\; & (s \to M(\alpha \times s)) \to \\ & (\alpha \to s \to M(\beta \times s)) \to \\ & s \to M(\beta \times s) \\ m \star k \;=\; & \lambda s.ms \star \lambda\langle x, s'\rangle.kxs' \end{aligned}$$

In this example M will be instantiated to the Powerset monad, and s to the type of lists of individuals. This transformation of the Powerset monad to provide State functionality allows such lists to be accessed and updated throughout semantic composition as

lists of discourse referents. To allow the indefinite *a linguist* to introduce a discourse referent, for example, Charlow defines the following operation, $(\cdot)^{\triangleright}$, for an underlying Powerset monad, though which we give for an arbitrary underlying monad M in the presence of State functionality:

$$(\cdot)^{\triangleright} : \mathsf{S}_T M\alpha \to \mathsf{S}_T M\alpha$$
$$m^{\triangleright} = m \star \lambda\langle x, s\rangle.\eta x(x :: s)$$

Here, the operation :: conses a new individual onto a list, thus providing it as a discourse referent. Now, one can associate the sentence *a linguist sleeps* with a discourse referent by having *a linguist* introduce it (given an updated instance of **FA**):

$$\mathbf{FA}(\lambda s.\{\langle x, s\rangle \mid \mathsf{ling}x\}^{\triangleright})(\eta\mathsf{sleep})$$
$$= \lambda s.\{\langle \mathsf{sleep}x, x :: s\rangle \mid \mathsf{ling}x\}$$

Thus the meaning of *a linguist* has changed, given our use of the State-transformed Powerset monad. The new meaning is, in fact, straightforward to obtain from the old meaning, however, in terms of a function lifting values from $M\alpha$ to $\mathsf{S}_T M\alpha$:

$$lift_{\mathsf{S}} : M\alpha \to \mathsf{S}_T M\alpha$$
$$lift_{\mathsf{S}}\, m = \lambda s.m \star \lambda x.\eta\langle x, s\rangle$$

It is in this sense that the addition of State functionality to meanings stated with respect to the Powerset monad is (in principle) *compositional*. Both the monadic combinators of the Powerset monad, *and* the meanings it is used to characterize, may be injected into the State setting.

Charlow uses this strategy to introduce analyses of quantificational noun phrases into the monadic setting. Taking inspiration from the continuation-style treatments of quantifiers of Barker (2002); Barker and Shan (2014), he employs the following Continuation monad transformer, C_T.

$$\mathsf{C}_T M\alpha = (\alpha \to Mt) \to Mt$$

$$\eta : \alpha \to (\alpha \to t) \to t$$
$$\eta a = \lambda c.ca$$

$$(\star) : ((\alpha \to Mt) \to Mt) \to$$
$$(\alpha \to (\beta \to Mt) \to Mt) \to$$
$$(\beta \to Mt) \to Mt$$
$$m \star k = \lambda c.m(\lambda x.kxc)$$

The underlying monad, in this case, is the State-transformed Powerset monad. Like the State monad transformer, the Continuation monad transformer also comes with a lifting function $lift_{\mathsf{C}}$:

$$lift_{\mathsf{C}} : M\alpha \to \mathsf{C}_T M\alpha$$
$$lift_{\mathsf{C}}\, m = \lambda k.m \star k$$

Now, a quantificational noun phrase such as *every philosopher* can be given the meaning $\lambda c, s.\{\langle \forall x.\mathsf{phil}x \to \exists y, s' : \langle True, s'\rangle \in cxs, s\rangle\}$.[3] Moreover, a sentence such as *every philosopher sees a linguist*, may be composed as follows, given a version of **FA** appropriate to the Continuation monad:

$$\begin{aligned}&\mathbf{FA}(\lambda c, s.\{\langle \forall x.\mathsf{phil}x \to \exists s' : \langle True, s'\rangle \in cxs, s\rangle\})\\ &\quad(\mathbf{FA}(\eta\mathsf{see}))(lift_{\mathsf{C}}(\lambda s.\{\langle y, s\rangle \mid \mathsf{ling}y\})))\\ &= \lambda c, s.\{\langle \forall x.\mathsf{phil}x \to \exists s' : \langle True, s'\rangle \in \bigcup_{\mathsf{ling}y} (c(\mathsf{see}yx)s), s\rangle\}\end{aligned}$$

Finally, as Charlow shows, such meanings of type $\mathsf{C}_T(\mathsf{S}_T\mathsf{P})t$ may be lowered to ones of type $\mathsf{S}_T\mathsf{P}t$ by applying them to the η of the State-transformed Powerset monad:

$$\begin{aligned}&lower_{\mathsf{C}}(\lambda c, s.\{\langle \forall x.\mathsf{phil}x \to \exists s' : \langle True, s'\rangle \in \bigcup_{\mathsf{ling}y} (c(\mathsf{see}yx)s), s\rangle\})\\ &= (\lambda c, s.\{\langle \forall x.\mathsf{phil}x \to \exists s' : \langle True, s'\rangle \in \bigcup_{\mathsf{ling}y} (c(\mathsf{see}yx)s), s\rangle\})\eta\\ &= \lambda s.\{\langle \forall x.\mathsf{phil}x \to \exists s' : \langle True, s'\rangle \in \{\langle \mathsf{see}yx, s\rangle \mid \mathsf{ling}y\}, s\rangle\}\\ &= \lambda s.\{\langle \forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx, s\rangle\}\\ &= \eta(\forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx)\end{aligned}$$

Such lowered meanings may, in turn, be lifted back into the Continuation monad, e.g., in order to further compose them with quantificational meanings:

$$\begin{aligned}&lift_{\mathsf{C}}(\eta(\forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx))\\ &= \lambda k.\eta(\forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx) \star k\\ &= \lambda k.k(\forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx)\\ &= \eta(\forall x.\mathsf{phil}x \to \exists y : \mathsf{ling}y \wedge \mathsf{see}yx)\end{aligned}$$

[3] The meanings Charlow provides for quantificational noun phrases headed by *every* are assembled in terms of more primitive operators which he defines elsewhere. We have thus somewhat simplified his presentation for our purposes.

Note that once a quantifier has been lowered, its scope is fixed. Thus Charlow composes $lift_C$ and $lower_C$ in this way, as an operator *reset*, in order to delimit the scope of quantifiers to finite clause boundaries. At the same time, as he shows, lowering does not affect the capacity of indefinite noun phrases and discourse referents to take scope; their side effects are still potent, as they are represented in terms of the State-transformed Powerset monad. As a consequence, the limited scopal possibilities for quantifiers and the flexible scoping behavior of indefinites and discourse referents may be modeled within the same continuation-based setting.

2.2 Monads and compositionality

The above discussion provides only a schematic presentation of the system of Charlow (2014). What we hope to have conveyed, however, is the manner in which the system, as a theory of indefiniteness, anaphora, and quantification, is monotonic and compositional in the senses introduced earlier. The theory of indefiniteness may be stated on its own, in terms of the Powerset monad, and then *embedded* into the combined theory of indefiniteness and anaphora, using a monad transformer. This combined theory may likewise be embedded into the combined theory of indefiniteness, anaphora, and quantification. To say that the embedding is monotonic and compositional is to say that it constitutes a (monad) homomorphism. Every lifting function *lift* has the property, in general, that it preserves the monadic combinators: $lift\,(\eta a) = \eta a$, and $lift\,(m \star k) = lift\,m \star \lambda x.lift\,(kx)$. Thus the theory stated with respect to the underlying monad is never truly forgotten and may, in fact, be used when convenient, i.e., before applying a *lift*.

What we aim to show in this paper is that the algebraic approach that we advocate has this property to an even greater degree. Indeed, the simple monadic approach requires, in many cases, determining a monad ahead of time that combines all of the effects which may occur in a given analysis. For example, say that one wants a theory of quantification on its own, independent of a theory of indefiniteness and anaphora. Then, one may employ the Continuation monad (as akin to Barker (2002); Barker and Shan (2014)); in this case, the definitions of η and $\star$ remain identical to those stated above, except for their types: the result type of the continuation is now simply t, so that $M\alpha = (\alpha \to t) \to t$. In turn, *every philosopher* may be given its usual generalized-quantifier meaning, i.e., $\lambda k.\forall x : \mathsf{phil}x \to kx$. Incorporating theories of indefiniteness and anaphora, however, will now prove more difficult. The State monad transformer

will provide a monad that takes a type α onto the type $s \to (\alpha \times s \to t) \to t$:

$$\begin{aligned} \mathsf{S}_T\mathsf{C}\alpha &= s \to \mathsf{C}(\alpha \times s) \\ &= s \to (\alpha \times s \to t) \to t \\ \eta a &= \lambda s, c.c\langle a, s\rangle \\ m \star k &= \lambda s, c.ms(\lambda\langle x, s'\rangle.kxs'c) \end{aligned}$$

Indeed, this result may appear, at first, to be suitable for a combined analysis of quantification and anaphora, but note, for example, that the value returned within the underlying Continuation monad will systematically have the type of a product. As a result, a *lower* operation will be required to have the type $(s \to (t \times s \to t) \to t) \to s \to t \times s$, but it is not obvious what the appropriate definition of such an operation would be.[4] Rather, in order to achieve the desired result, it seems that one must start with the Powerset monad, then incorporate anaphora, and then finally, incorporate quantification. Much more generally, the lifting functions $lift_X$ associated with each monad transformer X are often unidirectional, requiring that a choice of result *fixes* the underlying monad. Thus, ensuring that the resulting monad has a certain desired behavior will limit the flexibility with which one is able to combine different sources of functionality. In contrast, as we will show, the algebraic approach assigns a type with the minimum required effect to each meaning, and the combination of meanings with different effects systematically computes the appropriate result. There is thus no priority associated with one effect or another.[5]

A second difference between our algebraic approach and the monadic approach is that the types we compute for meanings exhibiting multiple effects is more informative: it yields a *linguistically meaningful* summary of the effects an expression gives rise to, as we will show.

[4]The most obvious candidate would be to throw out the state returned by the surrounding function on continuations — that is, such a *lower* would be defined as:

$$lower\, m = \lambda s.\langle m\, s\, (\lambda\langle a, s'\rangle.a), s\rangle$$

Such an operation, however, would invariably discard the anaphoric potential of its argument, treating, e.g., quantifiers and proper names alike. (In contrast, choosing State to be the underlying monad would allow anaphoric side effects to survive when they do not arise from bona fide quantifers.)

[5]One may wonder if the algebraic approach could be recast in terms of monad transformers. An issue which would arise is that the relevant lifting operation depends both on the meaning to which it is applied and on the context. Thus if we have n different atomic effects, we must consider $n \times (n-1)$ combinations of them (one for each pair of effects). Furthermore, monad transformers are ill-equipped to deal with effect bracketing, which we introduce in §3.5.

3 Algebraic effects via graded monads

As a way forward, we propose a double move: to simultaneously make the monadic approach modular and make its types more fine-grained.

First, we propose that semantic side effects be studied algebraically, in terms of equational laws characterizing the individual phenomena, which may then be combined. This move is inspired by Maršík and Amblard (2014, 2016); Maršík (2016), who develop a typed extension of the λ-calculus to study algebraic effects in semantics.[6] Unlike the approach of Maršík, we show how effects employed by semanticists — e.g., state and non-determinism — may be *recast* algebraically (while remaining in a pure setting), leading to more extensible grammars.

Second, we propose to track the relevant effects at the level of types, by using a graded monad.[7] In contrast to plain monads, graded monads are indexed with an abstraction of the effect that they perform, hereafter referred to simply as the "grade". The unit η of the monad is associated the unit grade (1). The grade of the composition of effects under $\star$ is the composition of their grades, written with the operator $(\cdot)$ (see Figure 1). Graded monads have been applied previously in the field of programming language theory to describe the semantics of algebraic effects (Katsumata, 2014; Mycroft et al., 2016; Orchard et al., 2019). In natural language semantics, they have been employed in the analysis of presupposition projection and anaphora (Grove, 2019). In our analysis, different phenomena are assigned different grades *independently of each other*. This means that the interpretations associated with individual phenomena may be freely composed, in order to yield grammars that combine the relevant effects.

One can then describe the interactions between effects using two sets of laws. The first set concerns the abstract level of grades. The second set concerns the concrete level of λ-terms and operations. These two sets of laws are related: any law between terms generates a corresponding law between grades; that is, any law governing terms is only allowable if there exists a corresponding law governing the behavior of grades. To illustrate, consider the unit and associativity laws on terms, which are part of the definition of a graded monad (Figure 1). For types to be preserved in the statement of associativity for $\star$, the $(\cdot)$ operator must be associative. Likewise, for types to be preserved in the identity laws regulating the behavior of η, 1 must be the left and right unit of $(\cdot)$. That is, grades must form a monoid.

[6] We trace the the idea of algebraic effects to the work of Plotkin and Power (2001); Plotkin and Pretnar (2008); Kiselyov and Ishii (2015).

[7] Our approach straightforwardly adapts to the setting of graded applicative functors (Kobele, 2018a). The two variants afford different dimensions of generalization.

Operations

$$\eta : \alpha \to \mathsf{M}_1\alpha$$
$$(\star) : \mathsf{M}_p\alpha \to (\alpha \to \mathsf{M}_q\beta) \to \mathsf{M}_{p\cdot q}\beta$$

Laws on terms

$$\eta\, v \star k = kv \qquad \text{(Left Identity)}$$
$$m \star \eta = m \qquad \text{(Right Identity)}$$
$$(m \star n) \star o = m \star (\lambda x.nx \star o) \qquad \text{(Associativity)}$$

Laws on grades

$$1 \cdot p = p \qquad \text{(Left Identity)}$$
$$p \cdot 1 = p \qquad \text{(Right Identity)}$$
$$(p \cdot q) \cdot r = p \cdot (q \cdot r) \qquad \text{(Associativity)}$$

Figure 1: Definition of a graded monad

From now on, we develop not just an *equational* theory of terms and grades, but a theory of *reduction*. That is, we use a reduction relation between terms written '$\longrightarrow$', and one between grades written '$\rightsquigarrow$'. These relations are the (respective) reflexive transitive congruence of the laws that we list below. By definition, two terms t_1 and t_2 are equal if they are inter-reducible; likewise for grades. At this point, our theory encompasses only the graded monad laws. At the introduction of any new law, we will ensure that the reduction relations on both grades and terms are confluent. In particular, we will ensure that the asserted laws are compatible with associativity; i.e., $g_1 \cdot (g_2 \cdot g_3)$ and $(g_1 \cdot g_2) \cdot g_3$ should always reduce to the same grade. Similarly at the level of terms: any proposed reduction rule should respect the Associativity law. We further discuss the importance of confluence in §5.1.

3.1 Compositional dynamic semantics

As recalled in §2, monadic semantics in the style of Shan (2002) aims to augment the interpretation of each syntactic category with an effect. In the present framework, this effect is graded. For example, if a sentence is interpreted as a truth value of type t in a non-effectful semantics, it is interpreted in our framework as a truth value associated

$$\frac{\langle u, m\rangle :: B/A \qquad \langle v, n\rangle :: A}{\langle uv, mn\rangle :: B}\,/ \qquad \frac{\langle u, m\rangle :: A \qquad \langle v, n\rangle :: A\backslash B}{\langle uv, nm\rangle :: B}\,\backslash$$

$$\frac{\langle u, m\rangle :: B/A \qquad \langle v, n\rangle :: A}{\langle uv, m \triangleright n\rangle :: B}\,/ \qquad \frac{\langle u, m\rangle :: A \qquad \langle v, n\rangle :: A\backslash B}{\langle uv, m \triangleleft n\rangle :: B}\,\backslash$$

$$\frac{\langle u, m\rangle :: A}{\langle u, m \star \lambda x.x\rangle :: A}\,\mu$$

Figure 2: Rules for forward and backward application

EXPRESSION	MEANING	CATEGORY	TYPE
john	$\eta\mathsf{j}$	NP	$\mathsf{M}_1 e$
walks	$\eta\mathsf{walk}$	$NP\backslash S$	$\mathsf{M}_1(e \to t)$

Table 1: A lexicon fragment

with an effect with some grade g, i.e., of type $\mathsf{M}_g t$.

Moreover, whereas in Montague semantics, one uses functional application, we additionally employ the graded applicative functor structure arising from the graded monad, characterized by (either of) the operators $(\triangleright)$ and $(\triangleleft)$:

$$\begin{aligned}(\triangleright) &: \mathsf{M}_p(\alpha \to \beta) \to \mathsf{M}_q\alpha \to \mathsf{M}_{p\cdot q}\beta\\ m \triangleright n &= m \star \lambda f.n \star \lambda x.\eta(fx)\\ (\triangleleft) &: \mathsf{M}_p\alpha \to \mathsf{M}_q(\alpha \to \beta) \to \mathsf{M}_{p\cdot q}\beta\\ m \triangleleft n &= m \star \lambda x.n \star \lambda f.\eta(fx)\end{aligned}$$

For illustration, we present a small applicative categorial grammar fragment in Table 1, and, in Figure 2, two rules of interpretation corresponding to functional application and two rules which make use of the applicative functor structure of our system. The need for seemingly redundant rules corresponding to simple functional application (above in Figure 2), in addition to applicative combination (below in Figure 2), arises from the fact that some meanings manipulate effectful values directly: their type is of the form $\mathsf{M}_p\alpha \to \mathsf{M}_q\beta$, rather than $\mathsf{M}_q(\alpha \to \beta)$. As such, they cannot be combined by either of the $\triangleright$ or $\triangleleft$ operators.[8] We additionally admit a rule (μ) which collapses a meaning of type $\mathsf{M}_{g_1}(\mathsf{M}_{g_2}\alpha)$ into one of type $\mathsf{M}_{g_1\cdot g_2}\alpha$ by sequencing it (via $\star$) with the identity function. In the following pages, we write '$\mu\, m$' in place of '$m \star \lambda x.x$'

[8]Indeed, the choice between the simple and applicative variants of $(/)$ in a derivational step is determined by the semantic types of the arguments being combined. Likewise for the choice between the variants of $(\backslash)$. (Semantic types of the form $\mathsf{M}_p\alpha \to \mathsf{M}_q\beta$ will be encountered in §3.5.) The same quirk justifies the presence of the μ rule, which we introduce next.

to be concise.

Using only the (\) rule, we may interpret *john walks* as a value whose grade is 1, i.e., one without any dynamic effect.

$$\frac{\langle \mathit{john}, \eta \mathsf{j} \rangle :: NP \qquad \langle \mathit{walks}, \eta \mathsf{walk} \rangle :: NP \backslash S}{\langle \mathit{john\ walks}, (\eta \mathsf{j}) \triangleleft (\eta \mathsf{walk}) \rangle :: S} \backslash$$

The definition of $(\triangleleft)$ and the monad laws allow this result to be reduced:

$$\begin{aligned} & (\eta \mathsf{j}) \triangleleft (\eta \mathsf{walk}) \\ = \ & \eta \mathsf{j} \star \lambda x.\eta \mathsf{walk} \star \lambda f.\eta (f\, x) \\ \longrightarrow \ & \eta (\mathsf{walk}\, \mathsf{j}) \qquad \text{(by Left Identity)} \end{aligned}$$

3.2 Anaphora

We can extend our analysis to account for anaphora. For any type α, we may posit a grade $\mathsf{Get}[d : \alpha]$, along with a new primitive, get_d:

$$\mathsf{get}_d : \mathsf{M}_{\mathsf{Get}[d:\alpha]} \alpha$$

The purpose of get_d is to retrieve a discourse referent d, whose type is α, from the linguistic context.[9] For instance, one can consider α to be e, the semantic type of *entities*, although any semantic type is supported, in principle.

The grade $\mathsf{Get}[d : \alpha]$ records that one *presupposes* the existence of a discourse referent with label d and type α. For example, get_d may be used to interpret a pronoun, with the typing $\mathsf{get}_d : \mathsf{M}_{\mathsf{Get}[d:e]} e$. The labels used for discourse referents are equipped with a decidable equality relation, but otherwise, they carry no meaning.[10] It should be noted that labels occur only inside grades — in §4, we show how the primitives may be interpreted into a label-free calculus. Finally, thanks to the typing rule for $\star$, a phrase which uses some number of discourse referents lists them all in its grade. For example, we might have the type $\mathsf{M}_{\mathsf{Get}[d_{masc}:e] \cdot \mathsf{Get}[d_{fem}:e]} t$ for the sentence *he likes her*.

Our goal is to formalize how grades interact. Since we do not keep track of the order in which discourse referents are introduced, we have the following equality on grades:

$$\mathsf{Get}[d_1 : \alpha] \cdot \mathsf{Get}[d_2 : \beta] = \mathsf{Get}[d_2 : \beta] \cdot \mathsf{Get}[d_1 : \alpha] \tag{1}$$

[9] We encode here roughly the notion of discourse referents of Karttunen (1976).

[10] This decision procedure tells whether or not there is co-reference. A possible implementation of it would be to match the properties of referents with predicates associated with anaphoric expressions (Bernardy et al., 2021).

Whenever we assert such a law on grades, it is important to check that it preserves the overall system's confluence in the presence of the other laws, including the monoid laws. So far, we have asserted only a commutation law, and it is easy to see that no problem arises.

Second, we do not keep track of *how many* references to a single discourse referent occur. Moreover, if two references to the same discourse referent are made, their types should agree. This is captured by the following law:[11]

$$\mathsf{Get}[d : \alpha] \cdot \mathsf{Get}[d : \alpha] \rightsquigarrow \mathsf{Get}[d : \alpha] \tag{2}$$

To complete the formal definition of the treatment of anaphoric expressions, it suffices to state how two instances of get_d should interact, as guided by the behavior of their grades. We employ two laws on terms (which we label according to the respective corresponding laws on grades):

$$\mathsf{get}_{d_1} \star \lambda x.\mathsf{get}_{d_2} \star \lambda y.\eta\langle x, y\rangle = \mathsf{get}_{d_2} \star \lambda y.\mathsf{get}_{d_1} \star \lambda x.\eta\langle x, y\rangle \tag{1'}$$

$$\mathsf{get}_d \star \lambda x.\mathsf{get}_d \star \lambda y.\eta\langle x, y\rangle \longrightarrow \mathsf{get}_d \star \lambda x.\eta\langle x, x\rangle \tag{2'}$$

The first law states that references to independent discourse referents commute. This law corresponds to law (1) on grades stating that the order of labels in a grade does not matter. The second law states that two references to the same discourse referent collapse to a single reference. This law corresponds to law (2) on grades, which collapses two associations with the same label. Note that, instead of first presenting the laws on grades, we could have stated the algebraic laws on terms and deduced their typing. Correct typing ensures that the behavior of terms, as captured by the algebraic laws, is mirrored by the behavior of grades, as captured by the grade laws.

[11] Our framework is, in principle, agnostic about the type system of the underlying λ-calculus. For instance, rich types, as proposed by Luo (2012), are supported, as is the simply typed λ-calculus. Even though we will avoid rich types in our analysis, we note that they may be particularly beneficial when it comes to tracking discourse referents. For instance, law (2) generalizes as follows:

$$\mathsf{Get}[d : \alpha] \cdot \mathsf{Get}[d : \beta] \rightsquigarrow \mathsf{Get}[d : \alpha \wedge \beta]$$

(where '$\alpha \wedge \beta$' refers to the meet of types α and β). Thus if two parts of a phrase refer to the same discourse referent, then the type associated with that discourse referent needs to be the meet of the types found in the parts.

Additionally, complex relations can be captured within the types of discourse referents. For example, the meaning of *john sees his dog* could be assigned the type $\mathsf{M}_{\mathsf{Get}([d:\Sigma(x:\mathsf{dog})(\mathsf{have}(\mathsf{j},x))])}$, which records a presupposition of the existence (via a Σ type) of John's dog. In the presence of rich types, one can additionally expect the types of the discourse referents to play a role in resolving anaphora.

EXPRESSION	MEANING	CATEGORY	TYPE
john	$(\eta \mathsf{j})^{\blacktriangleright}_d$	NP	$\mathsf{M}_{\mathsf{Put}[d:e]}e$
he	get_d	NP	$\mathsf{M}_{\mathsf{Get}[d:e]}e$
walks	$\eta\mathsf{walk}$	$NP\backslash S$	$\mathsf{M}_1(e \to t)$
sits	$\eta\mathsf{sit}$	$NP\backslash S$	$\mathsf{M}_1(e \to t)$
;	$\eta(\lambda\phi,\psi.\phi \wedge \psi)$	$S\backslash(S/S)$	$\mathsf{M}_1(t \to t \to t)\rangle$

Table 2: Adding discourse referents

3.3 Introducing discourse referents

As the dual to accessing discourse referents, one can introduce new ones. For this purpose, we add a new grade $\mathsf{Put}[d : \alpha]$, along with a new primitive:

$$\mathsf{put}_d : \alpha \to \mathsf{M}_{\mathsf{Put}[d:\alpha]}\diamond$$

The returned type, $\diamond$, is the unit type, thus signifying that put_d makes no significant contribution at the level of values. In terms of this primitive, one can define an operation $(\cdot)^{\blacktriangleright}_d$, which binds its argument to the discourse referent d. (The notation is inspired by the similar notation of Barker and Shan (2014), as well as of Charlow (2014).) This operation performs the dynamic effects associated with its argument, following which it binds the value returned to d:

$$(\cdot)^{\blacktriangleright}_d : \mathsf{M}_g\alpha \to \mathsf{M}_{g\cdot\mathsf{Put}[d:\alpha]}\alpha$$
$$m^{\blacktriangleright}_d = m \star \lambda x.\mathsf{put}_d x \star \lambda\diamond.\eta x$$

The '$\lambda\diamond$.' notation indicates that a value of type $\diamond$ is expected as an argument to the relevant λ-expression.

To illustrate, let us return to our running example, given the updated lexicon in Table 2. We now interpret *john walks* as follows.

$$\frac{\langle \mathit{john}, (\eta\mathsf{j})^{\blacktriangleright}_d\rangle :: NP \qquad \langle \mathit{walks}, \eta\mathsf{walk}\rangle :: NP\backslash S}{\langle \mathit{john\ walks}, (\eta\mathsf{j})^{\blacktriangleright}_d \triangleleft (\eta\mathsf{walk})\rangle :: S}\ \backslash$$

After unfolding the definitions and β-reducing, we obtain $\mathsf{put}_d\mathsf{j} \star \lambda\diamond.\eta(\mathsf{walk}\,\mathsf{j})$, whose type is $\mathsf{M}_{\mathsf{Put}[d:e]}t$, thus capturing that the discourse referent d has been introduced.

When considered on its own, put_d behaves similarly to get_d. The order of introduction does not matter:[12]

$$\mathsf{Put}[d_1 : \alpha] \cdot \mathsf{Put}[d_2 : \beta] = \mathsf{Put}[d_2 : \beta] \cdot \mathsf{Put}[d_1 : \alpha] \tag{3}$$

[12]Note that this algebra merely characterizes the *logic* of discourse referents, saying nothing about their accessibility from a cognitive standpoint.

Consequently, two discourse referents commute. We can formalize this as the following equation on terms:

$$\mathsf{put}_{d_1} a \star \lambda\diamond.\mathsf{put}_{d_2} b = \mathsf{put}_{d_2} b \star \lambda\diamond.\mathsf{put}_{d_1} a \tag{3'}$$

Although get_d and put_d arise independently and have interpretations on their own, we can describe their interactions in terms of algebraic laws. We illustrate this fact first on the relations on terms, by adding two laws:

$$\mathsf{put}_d a \star \lambda\diamond.\mathsf{get}_d \longrightarrow (\eta a)^{\blacktriangleright}_d \tag{4'}$$

$$\mathsf{put}_{d_1} a \star \lambda\diamond.\mathsf{get}_{d_2} \longrightarrow \mathsf{get}_{d_2} \star \lambda x.\mathsf{put}_{d_1} a \star \lambda\diamond.\eta x \qquad (d_1 \neq d_2) \tag{5'}$$

These laws ensure that get_d uses only the discourse referent d that put_d introduces. Assuming that the terms are well typed, the grades on the left should reduce to the grades on the right; consequently, the following laws hold on grades:

$$\mathsf{Put}[d : \alpha] \cdot \mathsf{Get}[d : \alpha] \rightsquigarrow \mathsf{Put}[d : \alpha] \tag{4}$$

$$\mathsf{Put}[d_1 : \alpha] \cdot \mathsf{Get}[d_2 : \beta] \rightsquigarrow \mathsf{Get}[d_2 : \beta] \cdot \mathsf{Put}[d_1 : \alpha] \qquad (d_1 \neq d_2) \tag{5}$$

The first law finds a satisfying linguistic justification: when a discourse referent is introduced, it is no longer presupposed. The second law ensures that introductions and uses of distinct discourse referents ignore each other.

To illustrate, consider composing the two utterances *john walks* with *he sits*. Given the lexicon in Table 2, this miniature discourse receives the following meaning:

$$\begin{aligned}
&[\![\textit{john walks; he sits}]\!] \\
&= ((\mathsf{put}_d \mathsf{j} \star \lambda\diamond.\eta(\mathsf{walk}\,\mathsf{j})) \triangleleft \eta(\lambda\phi,\psi.\phi \wedge \psi)) \triangleright (\mathsf{get}_d \star \lambda x.\eta(\mathsf{sit} x)) && \text{(by } /, \backslash, \mu\text{)} \\
&\longrightarrow (\mathsf{put}_d \mathsf{j} \star \lambda\diamond.\mathsf{get}_d) \star \lambda x.\eta(\mathsf{walk}\mathsf{j} \wedge \mathsf{sit} x) && \text{(by Associativity, Left Identity)} \\
&\longrightarrow (\eta \mathsf{j})^{\blacktriangleright}_d \star \lambda x.\eta(\mathsf{walk}\mathsf{j} \wedge \mathsf{sit} x) && \text{(by law (4))} \\
&\longrightarrow \mathsf{put}_d \mathsf{j} \star \lambda\diamond.\eta(\mathsf{walk}\mathsf{j} \wedge \mathsf{sit}\mathsf{j}) && \text{(by Associativity, Left Identity)}
\end{aligned}$$

The resulting meaning is of type $\mathsf{M}_{\mathsf{Put}[d:e]} t$; it introduces a discourse referent (d), but has no anaphoric presupposition, despite the presence of the pronoun *he*. That is, its reference is resolved. Checking confluence is a less easy exercise now than before. We can, however, convince ourselves that it holds by noting that the following reassociation is confluent:

$$\begin{aligned}
&\mathsf{Put}[d : \alpha] \cdot (\mathsf{Get}[d : \alpha] \cdot \mathsf{Get}[d : \alpha]) \\
\rightsquigarrow\ &\mathsf{Put}[d : \alpha] \cdot \mathsf{Get}[d : \alpha] && \text{(by law (2))} \\
\leftsquigarrow\ &(\mathsf{Put}[d : \alpha] \cdot \mathsf{Get}[d : \alpha]) \cdot \mathsf{Get}[d : \alpha] && \text{(by law (4))}
\end{aligned}$$

3.4 On the state monad

Both Charlow 2014 and other work in monadic dynamic semantics have employed the state monad, in order to model anaphora (Giorgolo and Unger, 2009; Unger, 2012). The foregoing formalization vindicates some of the state monad laws (laws (2) and (4)), but to get a full specification of the state monad, one additionally needs the following law:

$$\mathsf{get}_d \star \lambda x.\mathsf{put}_d x = \eta\diamond \tag{6'}$$

To preserve types, this law on terms requires the following law to hold on grades:

$$\mathsf{Get}[d : \alpha] \cdot \mathsf{Put}[d : \alpha] = 1 \tag{6}$$

Such a law is problematic, however, as it contravenes confluence:

$$\begin{aligned}
& \mathsf{Get}[d : \alpha] \cdot (\mathsf{Get}[d : \alpha] \cdot \mathsf{Put}[d : \alpha]) && \\
\rightsquigarrow\ & \mathsf{Get}[d : \alpha] \cdot 1 && \text{(by law (6))} \\
=\ & \mathsf{Get}[d : \alpha] && \text{(by Right Identity)} \\
\neq\ & 1 && \\
\rightsquigarrow\ & \mathsf{Get}[d : \alpha] \cdot \mathsf{Put}[d : \alpha] && \text{(by law (6))} \\
\rightsquigarrow\ & (\mathsf{Get}[d : \alpha] \cdot \mathsf{Get}[d : \alpha]) \cdot \mathsf{Put}[d : \alpha] && \text{(by law (2))}
\end{aligned}$$

Thus not all of the state monad laws can be imported into our framework, given how we employ graded types. What is responsible for this difference? The state monad is a theory of memory locations. According to the corresponding model of state, such memory locations pre-exist the lifetime of a program, and can be updated any number of times. Using a state monad to model anaphora would thus require that a constant set of referents be handled by the discourse. In comparison, our encoding of discourse referents is more precise: we record at the level of grades the exact discourse referents either introduced or presupposed. For our purposes, there is a fundamental difference between introducing a discourse referent and not introducing it. *A contrario*, we ought to reject the hypothetical law (6), which implies that using a discourse referent and then introducing it is, in fact, equivalent to doing nothing.

3.5 Quantification

As a further step, we may introduce another grade, Scope, in order to analyze expressions, such as *every*, which are commonly taken to denote generalized quantifier

meanings. Like those introduced above, this grade is accompanied by its own primitive:

$$\mathsf{scope} : ((e \to t) \to t) \to \mathsf{M}_{\mathsf{Scope}} e$$

Thus given a quantifier q of type $(e \to t) \to t$, $\mathsf{scope}\, q$ allows it to act as an entity at the level of values, i.e., in terms of the variable q binds, given that the primitive's return type is e.[13]

Indeed, the scopes of natural language quantifiers have been observed to be restricted in certain ways: one common view is that a quantifier cannot take scope outside the smallest finite clause in which it occurs syntactically. For example, *some cat fears every dog will chase it* can be understood only to imply the existence of a single highly pessimistic cat. To capture the effect of scope islands, we also introduce an operation $\langle\!\langle \cdot \rangle\!\rangle$ on grades and a primitive $\langle\!\langle \cdot \rangle\!\rangle$ which introduces it:

$$\langle\!\langle \cdot \rangle\!\rangle : \mathsf{M}_g t \to \mathsf{M}_{\langle\!\langle g \rangle\!\rangle} t$$

The intent is that $\langle\!\langle body \rangle\!\rangle$ allows one to ensure that a value bound in *body* using $\mathsf{scope}\, q$ is not available outside of *body*. This makes it possible to statically limit the scope of a variable bound by a quantifier.

The modularity provided by our approach allows us to import the laws regulating anaphora into the current setting. At the same time, we may describe the interactions between anaphora and quantification. To that end, we may state the following laws on grades:

$$\mathsf{Scope} \cdot \mathsf{Get}[d : \alpha] \rightsquigarrow \mathsf{Get}[d : \alpha] \cdot \mathsf{Scope} \tag{7}$$

$$\langle\!\langle 1 \rangle\!\rangle \rightsquigarrow 1 \tag{8}$$

$$\langle\!\langle g \cdot \mathsf{Scope} \rangle\!\rangle \rightsquigarrow \langle\!\langle g \rangle\!\rangle \tag{9}$$

$$\langle\!\langle g \cdot \mathsf{Scope} \cdot \mathsf{Put}[d : \alpha] \rangle\!\rangle \rightsquigarrow \langle\!\langle g \rangle\!\rangle \tag{10}$$

$$\langle\!\langle \mathsf{Put}[d : \alpha] \cdot g \rangle\!\rangle \rightsquigarrow \mathsf{Put}[d : \alpha] \cdot \langle\!\langle g \rangle\!\rangle \tag{11}$$

$$\langle\!\langle \mathsf{Get}[d : \alpha] \cdot g \rangle\!\rangle \rightsquigarrow \mathsf{Get}[d : \alpha] \cdot \langle\!\langle g \rangle\!\rangle \tag{12}$$

[13] The 'scope' notation is inspired by Maršík and Amblard (2016); Maršík (2016), though the constructors' exact purpose and semantics are different between the two approaches.

These laws are reflected on terms as follows:[14]

$$\begin{aligned}
& \mathsf{scope}\, q \star \lambda x.\mathsf{get}_d \star \lambda y.\eta\langle x, y\rangle \\
\longrightarrow\ & \mathsf{get}_d \star \lambda y.\mathsf{scope}\, q \star \lambda x.\eta\langle x, y\rangle && (7')\\
\langle\!\langle \eta\phi \rangle\!\rangle \longrightarrow\ & \eta\phi && (8')\\
& \langle\!\langle m \star \lambda x.\mathsf{scope}(qx) \star \lambda y.\eta(\phi xy)\rangle\!\rangle \\
\longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(qx(\lambda y.\phi xy))\rangle\!\rangle && (9')\\
& \langle\!\langle m \star \lambda x.\mathsf{scope}(qx) \star \lambda y.\mathsf{put}_d(axy) \star \lambda\diamond.\eta(\phi xy)\rangle\!\rangle \\
\longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(qx(\lambda y.\phi xy))\rangle\!\rangle && (10')\\
\langle\!\langle \mathsf{put}_d a \star k\rangle\!\rangle \longrightarrow\ & \mathsf{put}_d a \star \lambda\diamond.\langle\!\langle k\diamond\rangle\!\rangle && (11')\\
\langle\!\langle \mathsf{get}_d \star k\rangle\!\rangle \longrightarrow\ & \mathsf{get}_d \star \lambda x.\langle\!\langle kx\rangle\!\rangle && (12')
\end{aligned}$$

The occurrences of $\mathsf{Get}[d : \alpha]$ inside a bracket can be pulled to its left (laws (7) and (12)). Doing so, moreover, facilitates it meeting a $\mathsf{Put}[d : \alpha]$, which can then eliminate it.

Note that a law commuting $\mathsf{Put}[d : \alpha]$ and Scope is absent. Indeed, the grade $\mathsf{Scope} \cdot \mathsf{Put}[d : \alpha]$ corresponds to introducing an entity which may depend on another entity quantified over. Such a commutation should be rejected, as it would allow the introduced entity to escape its scope. An entity which is introduced inside a bracket, but before any Scope introduction, however, can be pulled out of the bracket, as per law (11).

A Scope introduced at the rightmost point of the body of a bracket can be reduced (law (9)): the operational interpretation of this law is to apply the quantifier to the returned property. If a discourse referent is introduced at the rightmost point of the body, immediately after Scope, then the introduction is simply ignored (law (10)). This should remain true for any number of introduced entities, moreover. To avoid introducing a scheme of reduction laws, we may use a law such as the following one, which coalesces indefinitely many introductions into one (or splits them) as needed:

$$\mathsf{Put}[d_1 : \alpha] \cdot \mathsf{Put}[d_2 : \beta] = \mathsf{Put}[\langle d_1, d_2\rangle \ :\ \alpha \times \beta] \tag{13}$$

$$\mathsf{put}_{d_1} a \star \lambda\diamond.\mathsf{put}_{d_2} b = \mathsf{put}_{\langle d_1, d_2\rangle}\langle a, b\rangle \tag{13'}$$

3.6 Indefinites

We now turn to indefinite noun phrases. Here, we pursue the idea of Charlow (2014, 2020a,b) that the meaning of an indefinite noun phrase is to non-deterministically

[14] We leave the proof of confluence to the reader. It relies on checking that appending something to the left-hand side of a reduction yields the same result as appending it to the right-hand side.

choose an entity from the set defined by its restriction. To do so, we introduce a new grade, $\mathsf{Choose}[\alpha]$, indexed by a type α, and associate it with the following primitive:

$$\mathsf{choose} \ : (\alpha \to t) \to \mathsf{M}_{\mathsf{Choose}[\alpha]}\alpha$$

We additionally provide the following law on grades:

$$\mathsf{Choose}[\alpha] \cdot \mathsf{Choose}[\beta] \rightsquigarrow \mathsf{Choose}[\alpha \times \beta] \tag{14}$$

This law is reflected at the level of terms as follows:

$$\begin{aligned} & \mathsf{choose}\ s_1 \star \lambda x.\mathsf{choose}\ (s_2 x) \star \lambda y.\eta\langle x, y\rangle \\ \longrightarrow\ & \mathsf{choose}\ (\lambda\langle x, y\rangle.s_1 x \wedge s_2 x y) \end{aligned} \tag{14$'$}$$

Intuitively, what this law says is that choosing two values in sequence is the same as choosing them simultaneously, as a pair. When it comes to the interaction with other grades, the $\mathsf{Choose}[\alpha]$ grade behaves similarly to $\mathsf{Put}[d : \alpha]$: it commutes to the left of $\mathsf{Put}[d : \alpha]$ (law (15)), but not to the left of $\mathsf{Get}[d : \alpha]$ or Scope; moreover, it is forgotten once it is sandwiched between Scope and the end of a bracket (laws (18) and (19)); however, it can nevertheless escape on the left of a bracket (law (17)).

$$\mathsf{Put}[d : \beta] \cdot \mathsf{Choose}[\alpha] \rightsquigarrow \mathsf{Choose}[\alpha] \cdot \mathsf{Put}[d : \beta] \tag{15}$$

$$\mathsf{Choose}[\alpha] \cdot \mathsf{Get}[d : \beta] \rightsquigarrow \mathsf{Get}[d : \beta] \cdot \mathsf{Choose}[\alpha] \tag{16}$$

$$\langle\!\langle \mathsf{Choose}[\alpha] \cdot g \rangle\!\rangle \rightsquigarrow \mathsf{Choose}[\alpha] \cdot \langle\!\langle g \rangle\!\rangle \tag{17}$$

$$\langle\!\langle g \cdot \mathsf{Scope} \cdot \mathsf{Choose}[\alpha] \rangle\!\rangle \rightsquigarrow \langle\!\langle g \rangle\!\rangle \tag{18}$$

$$\langle\!\langle g \cdot \mathsf{Scope} \cdot \mathsf{Choose}[\alpha] \cdot \mathsf{Put}[d : \beta] \rangle\!\rangle \rightsquigarrow \langle\!\langle g \rangle\!\rangle \tag{19}$$

To remain concise, we transcribe only the laws on terms that relate choose and scope (laws (18) and (19)).

$$\begin{aligned} & \langle\!\langle m \star \lambda x.\mathsf{scope}(qx) \star \lambda y.\mathsf{choose}\ (sxy) \star \lambda z.\eta(\phi xyz) \rangle\!\rangle \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(qx(\lambda y.\exists z : sxyz \wedge \phi xyz)) \rangle\!\rangle \end{aligned} \tag{18$'$}$$

$$\begin{aligned} & \langle\!\langle m \star \lambda x.\mathsf{scope}(qx) \star \lambda y.\mathsf{choose}\ (sxy) \star \lambda z.\mathsf{put}_d(axyz) \star \lambda \diamond.\eta(\phi xyz) \rangle\!\rangle \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(qx(\lambda y.\exists z : sxyz \wedge \phi xyz)) \rangle\!\rangle \end{aligned} \tag{19$'$}$$

To illustrate, consider the meaning derived for *every dog sees a cat*, given the updated

EXPRESSION	MEANING	CATEGORY	TYPE
dog	$\eta\mathsf{dog}$	N	$\mathsf{M}_1(e \to t)$
cat	$\eta\mathsf{cat}$	N	$\mathsf{M}_1(e \to t)$
sees	$\eta\mathsf{see}$	$(NP\backslash S)/NP$	$\mathsf{M}_1(e \to e \to t)$
a	$\eta(\lambda P^{e\to t}.\mathsf{choose}\ P)$	NP/N	$\mathsf{M}_1((e \to t) \to \mathsf{M}_{\mathsf{Choose}[e]}e)$
every	$\eta(\lambda P^{e\to t}.\mathsf{scope}(\mathsf{every}P)$	NP/N	$\mathsf{M}_1((e \to t) \to \mathsf{M}_{\mathsf{Scope}}e)$
ε	$\lambda\phi.\langle\!\langle\phi\rangle\!\rangle$	CP/S	$\mathsf{M}_g t \to \mathsf{M}_{\langle\!\langle g\rangle\!\rangle}t$

Table 3: Adding indefinites and quantifiers

lexicon in Table 3.

$$
\begin{aligned}
& [\![\textit{every dog sees a cat}]\!] \\
= \ & \langle\!\langle(\mu(\eta(\mathsf{scope}(\mathsf{every\ dog})))) \triangleleft ((\eta\mathsf{see}) \triangleright (\mu(\eta(\mathsf{choose\ cat}))))\rangle\!\rangle && \text{(by } /, \backslash, \mu\text{)} \\
\longrightarrow \ & \langle\!\langle(\mathsf{scope}(\mathsf{every\ dog})) \triangleleft ((\eta\mathsf{see}) \triangleright (\mathsf{choose\ cat}))\rangle\!\rangle && \text{(by Left Identity)} \\
\longrightarrow \ & \langle\!\langle\mathsf{scope}(\mathsf{every\ dog}) \star \lambda x.\mathsf{choose\ cat} \star \lambda y.\eta(\mathsf{see}yx)\rangle\!\rangle && \text{(by Left Identity)} \\
\longrightarrow \ & \langle\!\langle\eta(\mathsf{every\ dog}(\lambda x.\exists y : \mathsf{cat}y \wedge \mathsf{see}yx))\rangle\!\rangle && \text{(by law (18))} \\
\longrightarrow \ & \eta(\mathsf{every\ dog}(\lambda x.\exists y : \mathsf{cat}y \wedge \mathsf{see}yx)) && \text{(by law (8))}
\end{aligned}
$$

3.7 Determiners and donkey anaphora

The determiner algebra provides a new grade, Det, from which we define a new primitive, det, having the following type signature:

$$\mathsf{det} : ((e \to t) \to (e \to t) \to t) \to \mathsf{M}_{\mathsf{Det}}((e \to t) \to (e \to t) \to t)$$

det introduces a determiner meaning, which it merely returns. The utility of including determiners among the grades is manifest, however, when considering their interactions with other effects; in particular Choose$[\alpha]$:

$$\mathsf{Det} \cdot \mathsf{Get}[d : \alpha] \rightsquigarrow \mathsf{Get}[d : \alpha] \cdot \mathsf{Det} \tag{20}$$

$$\langle\!\langle g \cdot \mathsf{Det}\rangle\!\rangle \rightsquigarrow \langle\!\langle g\rangle\!\rangle \tag{21}$$

$$\langle\!\langle g \cdot \mathsf{Det} \cdot \mathsf{Put}[d : \alpha]\rangle\!\rangle \rightsquigarrow \langle\!\langle g\rangle\!\rangle \tag{22}$$

$$\langle\!\langle g \cdot \mathsf{Det} \cdot \mathsf{Choose}[e]\rangle\!\rangle \rightsquigarrow \langle\!\langle g\rangle\!\rangle \tag{23}$$

$$\langle\!\langle g \cdot \mathsf{Det} \cdot \mathsf{Choose}[e] \cdot \mathsf{Put}[d : \alpha]\rangle\!\rangle \rightsquigarrow \langle\!\langle g\rangle\!\rangle \tag{24}$$

Note that each of these laws has a corresponding law that involves Scope, rather than Det. Indeed, the corresponding laws on terms are analogous, except for laws (23) and (24), which are substantively different. Before we demonstrate this, we give the laws on terms for laws (21) and (22), which are realized by feeding a determiner meaning to its continuation:[15]

$$\begin{aligned} & \langle\!\langle m \star \lambda x.\mathsf{det}(Qx) \star \lambda Q'.\eta(\phi x Q') \rangle\!\rangle \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(\phi x(Qx)) \rangle\!\rangle && (21') \\ & \langle\!\langle m \star \lambda x.\mathsf{det}(Qx) \star \lambda Q'.\mathsf{put}_d(axQ') \star \lambda\diamond.\eta(\phi x Q') \rangle\!\rangle \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(\phi x(Qx)) \rangle\!\rangle && (22') \end{aligned}$$

More interesting are laws (23) and (24), each of which can be realized in two ways. The first gives rise to a "weak" existential reading of donkey sentences, while the second gives rise to a "strong" universal reading.[16] We provide the two laws corresponding to law (23), as those for law (24) are uninterestingly different (i.e., they additionally erase an occurrence of put_d).[17]

$$\begin{aligned} & \langle\!\langle m \star \lambda x.\mathsf{det}(Qx) \star \lambda Q'.\mathsf{choose}\ (sxQ') \star \lambda y.\eta(\phi x Q' y) \rangle\!\rangle && (23'\text{: weak}) \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(Qx(\lambda z.\exists y : sx(Qx)y \wedge \phi x(\lambda p, q.pz)y)(\lambda z.\exists y : sx(Qx)y \wedge \phi x(\lambda p, q.pz \wedge qz)y)) \rangle\!\rangle \end{aligned}$$

$$\begin{aligned} & \langle\!\langle m \star \lambda x.\mathsf{det}(Qx) \star \lambda Q'.\mathsf{choose}\ (sxQ') \star \lambda y.\eta(\phi x Q' y) \rangle\!\rangle && (23'\text{: strong}) \\ \longrightarrow\ & \langle\!\langle m \star \lambda x.\eta(Qx(\lambda z.\exists y : sx(Qx)y \wedge \phi x(\lambda p, q.pz)y)(\lambda z.\forall y : sx(Qx)y \rightarrow \phi x(\lambda p, q.pz \rightarrow qz)y)) \rangle\!\rangle \end{aligned}$$

With the lexicon in Table 4, we may derive the following meaning for *every new*

[15] We omit the corresponding law for law (20), which is uninterestingly different from its variant involving Scope.

[16] An alternative approach to rendering dynamically potent determiner meanings out of static ones of type $(e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$ is provided by Kobele (2018b). The laws of Kobele, which inspire ours, incorporate the contexts and discourse continuations of de Groote (2006) by relying on λ-homomorphisms.

[17] In this framework, these two laws are formally incompatible. We could add non-determinism, but prefer not to, in order to avoid obscuring our main points. In general, however, a full account will provide the conditions under which each reading is available; see, e.g., Kanazawa 1994.

EXPRESSION	MEANING	CATEGORY	TYPE
sees	$\eta\mathsf{see}$	$(NP\backslash S)/NP$	$\mathsf{M}_1(e \to t)$
pets	$\eta\mathsf{pet}$	$(NP\backslash S)/NP$	$\mathsf{M}_1(e \to t)$
new yorker	$\eta\mathsf{NYer}$	N	$\mathsf{M}_1(e \to t)$
dog	$\eta\mathsf{dog}$	N	$\mathsf{M}_1(e \to t)$
a	$\eta(\lambda P^{e\to t}.(\mathsf{choose}\ P)^{\blacktriangleright}_d)$	NP/N	$\mathsf{M}_1((e \to t) \to \mathsf{M}_{\mathsf{Choose}[e]\cdot\mathsf{Put}[d:e]}e)$
it	get_d	NP	$\mathsf{M}_{\mathsf{Get}[d:e]}e$
every	$\lambda P^{\mathsf{M}_g(e\to t)}.\mathsf{det\ every} \star \lambda Q.P \star \lambda P'.\mathsf{scope}(QP')$	NP/N	$\mathsf{M}_g(e \to t) \to \mathsf{M}_{\mathsf{Det}\cdot g\cdot\mathsf{Scope}}e$
who	$\eta(\lambda P, Q, x.Qx \wedge Px)$	$(N\backslash N)/(NP\backslash S)$	$\mathsf{M}_1((e \to t) \to (e \to t) \to e \to t)$
ε	$\lambda\phi.\langle\!\langle\phi\rangle\!\rangle$	CP/S	$\mathsf{M}_g t \to \mathsf{M}_{\langle\!\langle g\rangle\!\rangle} t$

Table 4: Adding determiners

yorker who sees a dog pets it:

$$
\begin{aligned}
& [\![\textit{every new yorker who sees a dog pets it}]\!] \\
= \ & \langle\!\langle (\mathsf{det\ every} \star \lambda Q. && \text{(by } /, \backslash, \mu\text{, Left Identity)} \\
& \quad (\mu(\eta(\mathsf{choose\ dog})^{\blacktriangleright}_d) \star \lambda x.\eta(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)) \\
& \quad\quad \star \lambda P.\mathsf{scope}(QP)) \triangleleft ((\eta\mathsf{pet}) \triangleright \mathsf{get}_d)\rangle\!\rangle \\
\longrightarrow \ & \langle\!\langle \mathsf{det\ every} \star \lambda Q. && \text{(by Left Identity)} \\
& \quad ((\mathsf{choose\ dog})^{\blacktriangleright}_d \star \lambda x.\eta(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)) \\
& \quad\quad \star \lambda P.\mathsf{scope}(QP) \star \lambda y.\mathsf{get}_d \star \lambda x'.\eta(\mathsf{pet}x'y)\rangle\!\rangle \\
\longrightarrow \ & \langle\!\langle \mathsf{det\ every} \star \lambda Q. && \text{(by law (7))} \\
& \quad ((\mathsf{choose\ dog})^{\blacktriangleright}_d \star \lambda x.\eta(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)) \\
& \quad\quad \star \lambda P.\mathsf{get}_d \star \lambda x'.\mathsf{scope}(QP) \star \lambda y.\eta(\mathsf{pet}x'y)\rangle\!\rangle \\
\longrightarrow \ & \langle\!\langle \mathsf{det\ every} \star \lambda Q.(\mathsf{choose\ dog})^{\blacktriangleright}_d && \text{(by Associativity, Left Identity)} \\
& \quad\quad \star \lambda x.\mathsf{get}_d \star \lambda x'.\mathsf{scope}(Q(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)) \star \lambda y.\eta(\mathsf{pet}x'y)\rangle\!\rangle \\
\longrightarrow \ & \langle\!\langle \mathsf{det\ every} \star \lambda Q.\mathsf{choose\ dog} && \text{(by Associativity, Left Identity, law (4))} \\
& \quad\quad \star \lambda x.\mathsf{put}_d x \star \lambda\diamond.\mathsf{scope}(Q(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)) \star \lambda y.\eta(\mathsf{pet}xy)\rangle\!\rangle \\
\longrightarrow \ & \langle\!\langle \mathsf{det\ every} \star \lambda Q.\mathsf{choose\ dog} && \text{(by law (9))} \\
& \quad\quad \star \lambda x.\mathsf{put}_d x \star \lambda\diamond.\eta(Q(\lambda y.\mathsf{NYer}y \wedge \mathsf{see}xy)(\lambda y.\mathsf{pet}xy))\rangle\!\rangle
\end{aligned}
$$

At this point, we have two options, depending on the reduction rule we choose to coincide with law (24). If we opt for the weak reading, we can continue as follows:

$$
\begin{aligned}
\longrightarrow \ & \langle\!\langle \eta(\mathsf{every}(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy))(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy \wedge \mathsf{pet}xy)\rangle\!\rangle \\
\longrightarrow \ & \eta(\mathsf{every}(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy))(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy \wedge \mathsf{pet}xy)
\end{aligned}
$$

On this reading, every New Yorker who sees a dog pets at least one dog they see. If

we opt instead for the strong reading, we can continue as follows:

$$\longrightarrow \langle\!\langle \eta(\mathsf{every}(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy))(\lambda y.\forall x : \mathsf{dog}x \to ((\mathsf{NYer}y \wedge \mathsf{see}xy) \to \mathsf{pet}xy)) \rangle\!\rangle$$
$$\longrightarrow \eta(\mathsf{every}(\lambda y.\exists x : \mathsf{dog}x \wedge \mathsf{NYer}y \wedge \mathsf{see}xy))(\lambda y.\forall x : \mathsf{dog}x \to ((\mathsf{NYer}y \wedge \mathsf{see}xy) \to \mathsf{pet}xy))$$

Now, every New Yorker who sees a dog pets every dog they see; i.e., the reading attributed to donkey sentences by most dynamic semantic accounts.

4 Realization in terms of a pure calculus

In this section, we provide meanings to the grades, the operations, and their relation in terms of the simply typed λ-calculus with products (hereafter, STLC). We will only provide proof sketches here, but we note that the contents of this section and §3 have been formalized using the Agda proof assistant.

Theorem 1 (Coherence of reduction relations)**.** If $t_1 : \mathsf{M}_{g_1}\alpha$, $t_2 : \mathsf{M}_{g_2}\alpha$, and $t_1 \longrightarrow t_2$, then $g_1 \rightsquigarrow g_2$.

Proof. By case analysis. □

Definition 1 (Interpretation of grades)**.** For every graded type $\mathsf{M}_g\alpha$, there is a semantic interpretation $[\![\mathsf{M}_g\alpha]\!] = S_g([\![\alpha]\!])$ as a type in the STLC (or, more generally, in the underlying typed λ-calculus without effects). $[\![\cdot]\!]$ preserves STLC types and is defined on graded types as follows.

$$\begin{aligned}
S_1(\alpha) &= \alpha \\
S_{f\cdot g}(\alpha) &= S_f(S_g(\alpha)) \\
S_{\mathsf{Put}[d:\beta]}(\alpha) &= [\![\beta]\!] \times \alpha \\
S_{\mathsf{Get}[d:\beta]}(\alpha) &= [\![\beta]\!] \to \alpha \\
S_{\mathsf{Scope}}(\alpha) &= ((e \to t) \to t) \times (e \to \alpha) \\
S_{\mathsf{Choose}[\gamma]}(\alpha) &= ([\![\gamma]\!] \to t) \times ([\![\gamma]\!] \to \alpha) \\
S_{\mathsf{Det}}(\alpha) &= ((e \to t) \to (e \to t) \to t) \times (((e \to t) \to (e \to t) \to t) \to \alpha) \\
S_{\langle g \rangle}(\alpha) &= S_g(t) \times (t \to \alpha)
\end{aligned}$$

We stress that this interpretation is entirely modular in the sense that the meanings of the atomic effects are devised independently, without taking into account any interplay between effects. (It is a homomorphism on the grade structure.) As a rule,

if the primitive operation associated with an effect takes as input an object of type X, then we take the product with X in the interpretation. Conversely, if such a primitive returns a type Y, then Y is found as the domain of an arrow in the interpretation. A consequence of this modularity is that all the results of this section can be proven in a modular fashion, by case analysis for each atomic grade. For grade composition, a straightforward induction applies.

Lemma 1. S is a graded monad.

The proof relies on the following facts: 1) each atomic grade is interpreted as a functor; 2) the unit grade is interpreted as the identity functor; 3) the composition of grades is interpreted as functor composition.

Theorem 2. If $g_1 \rightsquigarrow g_2$, then there is a function $f : S_{g_1}(\alpha) \to S_{g_2}(\alpha)$ for each STLC type α.

Proof. This is a constructive proof done by case analysis. The function f says how (the semantic interpretations of) effects are transformed by reductions. For instance, the law

$$\mathsf{Put}[d : \alpha] \cdot \mathsf{Get}[d : \alpha] \rightsquigarrow \mathsf{Put}[d : \alpha]$$

corresponds to functions $f : (\alpha \times (\alpha \to \beta)) \to (\alpha \times \beta)$, which pass the newly introduced value (of type α) to its continuation, which then uses it. That is, $f\langle x, k\rangle = \langle x, kx\rangle$. □

We call the relation induced by such functions '$[\![g_1 \rightsquigarrow g_2]\!]$'. (That is, $x \,[\![g_1 \rightsquigarrow g_2]\!]\, y$ iff $fx = y$, where f is a function provided by Theorem 2.)[18] Finally, it bears repeating that the above construction *defines the semantics* of the reduction relation, and is thus the keystone of the interpretation.

Definition 2 (Interpretation of terms)**.** For every well-typed term $t : \mathsf{M}_g\alpha$, we define an interpretation $[\![t]\!]$ such that $[\![t]\!] : S_g([\![\alpha]\!])$. The interpretations of η and $\star$ are given by the graded monadic structure of S (Lemma 1). The recipe for interpreting each atomic grade is based straightforwardly on the type of the primitive giving rise to the grade. For example, $[\![\mathsf{get}_d]\!] = \lambda x.x$, $[\![\langle\!\langle t \rangle\!\rangle]\!] = \langle [\![t]\!], \lambda\phi.\phi\rangle$, etc.

[18] In addition, any two reduction functions associated with grade equality form an isomorphism; i.e., if $g_1 = g_2$, then $S_{g_1}(\alpha) \cong S_{g_2}(\alpha)$ (for any α). This can be seen by noting two facts. First, that the monoid laws regulating grades give rise to identity functions on terms, since 1 is interpreted via the identity functor, and functor composition is associative. Second, that otherwise equivalent grades require merely rearranging either the order of λ-abstractions or the components of a tuple in the interpretation. For example, $\mathsf{Put}[d_1 : \alpha] \cdot \mathsf{Put}[d_2 : \beta] = \mathsf{Put}[d_2 : \beta] \cdot \mathsf{Put}[d_1 : \alpha]$ corresponds to the isomorphisms $[\![\alpha]\!] \times ([\![\beta]\!] \times \gamma) \cong [\![\beta]\!] \times ([\![\alpha]\!] \times \gamma)$.

Theorem 3 (Adequacy of the interpretation)**.** The interpretation of terms respects the interpretation of grades and the interpretation of reductions as functions. Formally, if $t_1 : \mathsf{M}_{g_1}\alpha$, $t_2 : \mathsf{M}_{g_2}\alpha$, and $t_1 \longrightarrow t_2$, then $[\![t_1]\!][\![g_1 \rightsquigarrow g_2]\!][\![t_2]\!]$.

This theorem essentially tells us that the axiomatization of term reductions exactly fits the interpretations of grades. As a result, if one wishes, one may omit the axiomatization, and use only the interpretation and the corresponding reduction relation. We have chosen to present the axiomatic view to emphasize the operational behavior of terms having effects. If one is interested only in the end product (i.e., pure λ-terms), then one would be better off axiomatizing grades and their relations only. This way, by omitting the axiomatization of operations and algebraic laws, one can describe their compositional meanings (as in Definitions 1 and 2) directly.

5 Related work

5.1 Effects and handlers

To improve compositionality, general *effects and handlers* systems have been proposed for dynamic semantics by Maršík and Amblard (2014, 2016); Maršík (2016). In these approaches, new operations, such as get or put, can be declared and defined locally in terms of the ambient calculus. These approaches have much in common with ours, insofar as they provide modular interpretations of the effectful operations they employ. Furthermore, while effectful meanings are defined in a typed extension of λ-calculus, they yield terms of a pure λ-calculus once they are handled.

The chief difference between the effects and handlers approach and the one advanced here, which makes algebraic laws central, is that the former approach demands that every occurrence of an operation be interpreted (i.e., *handled*) independently of the context in which it occurs. This requirement enforces absolute compositionality of interpretation, whereas our method does not. In other words, while our *syntax* is compositional, the *eventual* interpretation of a grade may depend on its context. Indeed, our reduction rules are written so that the meaning of an operation can depend on its neighbors. This design allows the interpretation of Scope, for example, to occur only at the rightmost point in a bracket, where it may receive a function of type $e \to t$. Crucially, nevertheless, the results yielded by the applications of laws are compositional: due to associativity and confluence, one may safely apply reduction rules to a term m or a grade g independently of the context in which m or g occurs. When combining m with a continuation k, it suffices to consider their reduced forms: confluence guarantees that the result of $m \star k$ is the same, regardless of what reductions occur before

their combination.

5.2 The underlying calculus

Even though we have assumed the STLC as our ambient calculus, monadic and algebraic effects approaches (and, more generally, approaches based on computational effects) are agnostic as to the type system used by the underlying λ-calculus, be it Martin Löf Type Theory (Martin-Löf, 1984) or one of its variants, System F (Girard, 1972), Cooper's TTR (Cooper and Ginzburg, 2015), Asher's TCL (Asher, 2011), etc. Thus our approach (as others) may be added to such systems without modifying the respective calculi.

5.3 Graded effects

Our treatment of discourse phenomena in terms of grades is partially inspired by the interpretation of Cooper storage in terms of a graded applicative functor due to Kobele (2018a). Kobele employs grades that correspond to stores of quantifier meanings, in order to encode the types of both stored quantifiers and the variables they bind. We employ somewhat richer grades than Kobele, in order to encode, e.g., discourse referents. Such rich grades allow us to describe linguistically meaningful interactions at the level of types that reflect the algebraic laws that apply at the level of terms.

5.4 Modalities instead of graded monads

Our presentation relies on the standard structure of λ-calculi to encode dynamic effects as monads. This causes a certain amount of notational weight in the axiomatization. Namely, we have to use a family of operators $\triangleright$, $\triangleleft$, $\star$, etc., instead of simple functional application.

To avoid this overhead, an alternative presentation could use modalities to represent the combination of dynamic effects associated with a value. Several calculi supporting these kind of modalities have been developed recently (Petricek et al., 2014; Orchard et al., 2019; Abel and Bernardy, 2020).

6 Conclusion

We have proposed a framework which both unifies and refines approaches to dynamic semantics based on monads. The key idea is to break down effects into atomic *grades*. The interactions among grades are provided by algebraic laws, which can be presented

in a modular fashion. Even though the number of possible laws grows quadratically with the number of possible effects, laws are much fewer than this theoretical maximum if we exclude the mechanical commutation laws.

The process of applying this refinement reveals possible improvements to earlier analyses, for example regarding the interpretation of anaphora using the state monad (§3.4). The use of a bracketing operation to delimit scope appears to be new, and is an essential device in the interpretation of quantification effects.

Our framework can either be given a purely axiomatic treatment (§3), or, like many accounts, be provided as part of a pure λ-calculus (§4). In future work, we intend to describe more effects within the same framework, including presupposition and conventional implicature.

References

Abel, Andreas, and Jean-Philippe Bernardy. 2020. A unified view of modalities in type systems. *Proceedings of the ACM on Programming Languages* 4:90:1–90:28. https://doi.org/10.1145/3408972.

Asher, Nicholas. 2011. *Lexical Meaning in Context: A Web of Words*. Cambridge: Cambridge University Press. https://www.cambridge.org/core/books/lexical-meaning-in-context/F1D01632AD5B491A94860A350B9E764A.

Barker, Chris. 2002. Continuations and the Nature of Quantification. *Natural Language Semantics* 10:211–242. https://doi.org/10.1023/A:1022183511876.

Barker, Chris, and Chung-chieh Shan. 2014. *Continuations and natural language*, volume 53. Oxford studies in theoretical linguistics.

Bernardy, Jean-Philippe, Stergios Chatzikyriakidis, and Aleksandre Maskharashvili. 2021. A Computational Treatment of Anaphora and Its Algorithmic Implementation. *Journal of Logic, Language and Information* 30:1–29. https://doi.org/10.1007/s10849-020-09322-7.

Charlow, Simon. 2014. On the semantics of exceptional scope. PhD Thesis, NYU, New York. https://semanticsarchive.net/Archive/2JmMWRjY.

Charlow, Simon. 2020a. The scope of alternatives: indefiniteness and islands. *Linguistics and Philosophy* 43:427–472. https://doi.org/10.1007/s10988-019-09278-3.

Charlow, Simon. 2020b. Static and dynamic exceptional scope. https://ling.auf.net/lingbuzz/004650, publisher: Rutgers University Published: LingBuzz.

Cooper, Robin, and Jonathan Ginzburg. 2015. Type Theory with Records for Natural Language Semantics*. In *The Handbook of Contemporary Semantic Theory*, 375–407. John Wiley & Sons, Ltd. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118882139.ch12, section: 12 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118882139.ch12.

Giorgolo, Gianluca, and Ash Asudeh. 2012. $\langle$M, η, $\star\rangle$ Monads for Conventional Implicatures. In *Proceedings of Sinn und Bedeutung 16*, ed. Ana Aguilar Guevara, Anna Chernilovskaya, and Rick Nouwen, MITWPL, 265–278. http://mitwpl.mit.edu/open/sub16/Giorgolo.pdf.

Giorgolo, Gianluca, and Christina Unger. 2009. Coreference without Discourse Referents. In *Proceedings of the 19th Meeting of Computational Linguistics in the Netherlands*, ed. Barbara Plank, Erik Tjong Kim Sang, and Tim Van de Cruys, 69–81.

Girard, Jean-Yves. 1972. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Doctoral Dissertation, Université Paris 7.

de Groote, Philippe. 2001. Type raising, continuations, and classical logic. In *Proceedings of the 13th Amsterdam Colloquium*, ed. Robert van Rooij and Martin Stokhof, 97–101. Institute for Logic, Language and Computation, Universiteit van Amsterdam.

de Groote, Philippe. 2006. Towards a Montagovian Account of Dynamics. *Semantics and Linguistic Theory* 16:1–16. https://journals.linguisticsociety.org/proceedings/index.php/SALT/article/view/2952, number: 0.

Grove, Julian. 2019. Scope-taking and presupposition satisfaction. PhD Thesis, University of Chicago, Chicago. https://semanticsarchive.net/Archive/TRmOTkzM.

Kanazawa, Makoto. 1994. Weak vs. strong readings of donkey sentences and monotonicity inference in a dynamic setting. *Linguistics and Philosophy* 17:109–158. https://doi.org/10.1007/BF00984775.

Karttunen, Lauri. 1976. Discourse referents. In *Notes from the Linguistic Underground*, volume 7 of *Syntax and Semantics*. New York: Academic Press. http://web.stanford.edu/~laurik/publications/archive/discref.pdf.

Katsumata, Shin-ya. 2014. Parametric effect monads and semantics of effect systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, 633–645. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2535838.2535846.

Kiselyov, Oleg, and Hiromi Ishii. 2015. Freer monads, more extensible effects. *ACM SIGPLAN Notices* 50:94–105. https://doi.org/10.1145/2887747.2804319.

Kobele, Gregory M. 2018a. The Cooper Storage Idiom. *Journal of Logic, Language and Information* 27:95–131. https://doi.org/10.1007/s10849-017-9263-1.

Kobele, Gregory M. 2018b. Modularizing semantics. https://home.uni-leipzig.de/gkobele/files/slides/Kobele18Frankfurt.pdf.

Luo, Zhaohui. 2012. Common Nouns as Types. In *Logical Aspects of Computational Linguistics*, ed. Denis Béchet and Alexander Dikovsky, Lecture Notes in Computer Science, 173–185. Berlin, Heidelberg: Springer.

Martin-Löf, Per. 1984. *Intuitionistic type theory*. Napoli: Bibliopolis. https://archive-pml.github.io/martin-lof/pdfs/Bibliopolis-Book-retypeset-1984.pdf.

Maršík, Jirka, and Maxime Amblard. 2016. Introducing a Calculus of Effects and Handlers for Natural Language Semantics. In *Formal Grammar*, ed. Annie Foret, Glyn Morrill, Reinhard Muskens, Rainer Osswald, and Sylvain Pogodalla, Lecture Notes in Computer Science, 257–272. Berlin, Heidelberg: Springer.

Maršík, Jiří. 2016. Effects and handlers in natural language. phdthesis, Université de Lorraine. https://hal.inria.fr/tel-01417467.

Maršík, Jiří, and Maxime Amblard. 2014. Algebraic Effects and Handlers in Natural Language Interpretation. In *Natural Language and Computer Science*, ed. Valeria de Paiva, Walther Neuper, Pedro Quaresma, Christian Retoré, Lawrence S. Moss, and Jordi Saludes, volume TR 2014-002 of *Joint Proceedings of the Second Workshop on Natural Language and Computer Science (NLCS'14) & 1st International Workshop on Natural Language Services for Reasoners (NLSR 2014)*. Vienne, Austria: Center for Informatics and Systems of the University of Coimbra. https://hal.archives-ouvertes.fr/hal-01079206.

Mycroft, Alan, Dominic Orchard, and Tomas Petricek. 2016. Effect Systems Revisited—Control-Flow Algebra and Semantics. In *Semantics, Logics, and Calculi: Essays Dedicated to Hanne Riis Nielson and Flemming Nielson on the Occasion of Their*

60th Birthdays, ed. Christian W. Probst, Chris Hankin, and René Rydhof Hansen, Lecture Notes in Computer Science, 1–32. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-27810-0_1.

Orchard, Dominic, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages* 3:110:1–110:30. https://doi.org/10.1145/3341714.

Petricek, Tomas, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*, ICFP '14, 123–135. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2628136.2628160.

Plotkin, Gordon, and Matija Pretnar. 2008. A Logic for Algebraic Effects. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, 118–129. ISSN: 1043-6871.

Plotkin, Gordon D., and John Power. 2001. Adequacy for Algebraic Effects. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS '01, 1–24. Berlin, Heidelberg: Springer-Verlag.

Shan, Chung-chieh. 2002. Monads for natural language semantics. *arXiv:cs/0205026* http://arxiv.org/abs/cs/0205026, arXiv: cs/0205026.

Unger, Christina. 2012. Dynamic Semantics as Monadic Computation. In *New Frontiers in Artificial Intelligence*, ed. Manabu Okumura, Daisuke Bekki, and Ken Satoh, Lecture Notes in Computer Science, 68–81. Berlin, Heidelberg: Springer.