1 Overview

Continuized **C**ombinatory **C**ategorial **G**rammars supplement traditional applicative categorial grammars with a small number of combinators that allow expressions to take scope over their compositional contexts (e.g. Shan & Barker 2006, Barker & Shan 2008, 2014). Though CCCGs offer robust accounts of in-scope binding (ibid.), the treatment of dynamic binding (i.e. cross-sentential and donkey anaphora) proposed in Barker & Shan 2008 overgenerates (Charlow 2010; see Barker & Shan 2014 for in-depth discussion). Conversely, the related theory of de Groote 2006 (which relies on continuations, but is not a CCG) handles dynamic binding elegantly, but does not offer any general account of scope-taking. In addition, both Barker & Shan 2008 and de Groote 2006 analyze dynamic binding by indefinites, but not by expressions such as *exactly one linguist*, which differs in important ways (cf. Kamp & Reyle 1993).

This paper presents an explicit CCCG account of dynamic binding inspired by the computer science notion of a *monad* (e.g. Moggi 1989, Wadler 1992, 1994, 1995, Shan 2002). Monads were developed as an abstraction for structuring extensions to the pure lambda calculus; they enable us to modularly add *side effects* (cf. Shan 2005) to a programming language or natural language grammar. It is proposed that dynamic binding can be captured by recognizing two kinds of linguistic side effects: **state** and **nondeterminism**. State corresponds to the ability to introduce discourse referents, while nondeterminism means the ability to track multiple computations in parallel, which facilitates a referential treatment of indefinite expressions.

Concretely, a monad determines a pair of combinators $\langle \eta, \star \rangle$, one for wrapping values inside trivially effectful structures, and another for composing programs that may incur side effects. Our proposal is to replace the standard lifting and lowering operations of CCCGs with (respectively) $\star$ and η. Combined with some intuitive lexical entries, this has the automatic effect of producing a CCCG that countenances side effects. Crucially, given the underlying continuations-based scaffolding, the problem of integrating well-motivated combinatorial approaches to scope with insights from dynamic semantics reduces to finding a monad for state and nondeterminism. Though we focus on the case of dynamic binding here, any system of effects can be grafted onto a continuized CCG in this way.

2 Continuations

Simplifying somewhat,[1] we take it that a standard CCCG consists of three polymorphic combinators, **Lift**, **Triv**, and **Scope**, in addition to the basic categorial slashes that encode forward and backward function application. These combinators are defined in Figure 1a (along with a characterization of the slashes' behavior). The inference rules make use of a type constructor K parameterized by two concrete types. Formally, $\mathsf{K}\,a\,r ::= r/(r/a)$; the r/a argument to something of type $\mathsf{K}\,a\,r$ is called its *continuation*, and represents the denotation of its context. Intuitively, expressions with type $\mathsf{K}\,a\,r$ behave like things of type a within compositional contexts of type r, the 'result type' of the computation. For example, extensional generalized quantifiers have type $\mathsf{K}\,e\,t$, since they behave locally as individuals but quantify over constituents of type t.

As for semantics, **Lift** turns any expression into a function on its continuation; it is a polymorphic generalization of the familiar lifting procedure that converts individuals to the corresponding principal ultrafilters (cf. Montague 1974). **Scope** defines the compositional process whereby two continuized expressions are combined, giving the left scope over the right. Finally, **Triv** is a trivial continuation (the identity function) used to delimit contexts of evaluation and expose underlying semantic values. Shan & Barker 2006 show that these three operations suffice to derive cases of inverse scope (that is, where an expression scopes over something to its left).

3 Adding side effects

Continuized grammars need not place inherent restrictions on r, the type of the context. For simple cases where only boolean scope-taking is at issue (cf. Barwise & Cooper 1981), we can set $r ::= t$. But in general, we may be interested in contexts that denote in richer semantic spaces, as is typically the case with questions, focus, attitude reports, dynamic updates, etc. In such cases, we enrich the return type to match the phenomenon of interest.

We propose to model the sorts of semantic enrichments motivated by these phenomena using monads (Wadler 1994, Liang et al. 1995), which serve much the same purpose in capturing the semantics of programming language *side effects* like IO. A monad is a triple $\langle \mathsf{M}, \eta, \star \rangle$ of a type construc-

1 In particular, abstracting away from the multimodal presentation of Shan & Barker 2006, Barker & Shan 2008, 2014.

$$\frac{\Gamma \vdash f : b/a \quad \Delta \vdash e : a}{\Gamma\cdot\Delta \vdash f\,e : b}\,/ \qquad \frac{\Delta \vdash e : a \quad \Gamma \vdash f : a\backslash b}{\Delta\cdot\Gamma \vdash f\,e : b}\,\backslash$$

$$\frac{}{\varepsilon \vdash \lambda x.\,x : a/a}\,\textsf{Triv} \qquad \frac{}{\varepsilon \vdash \lambda x k.\,k\,x : \mathsf{K}\,a\,r/a}\,\textsf{Lift} \qquad \frac{}{\varepsilon \vdash \mathbf{S} : (\mathsf{K}\,b\,r/\mathsf{K}\,a\,r)/\mathsf{K}\,(b/a)\,r}\,\textsf{Scope}$$

(a) Continuized CCG without side effects, fixing a result type r.

$$\frac{\Gamma \vdash f : b/a \quad \Delta \vdash e : a}{\Gamma\cdot\Delta \vdash f\,e : b}\,/ \qquad \frac{\Delta \vdash e : a \quad \Gamma \vdash f : a\backslash b}{\Delta\cdot\Gamma \vdash f\,e : b}\,\backslash$$

$$\frac{}{\varepsilon \vdash \eta : \mathsf{M}\,a/a}\,\eta \qquad \frac{}{\varepsilon \vdash (\star) : \mathsf{K}\,a\,\mathsf{M}\,r/\mathsf{M}\,a}\,\star \qquad \frac{}{\varepsilon \vdash \mathbf{S} : (\mathsf{K}\,b\,r/\mathsf{K}\,a\,r)/\mathsf{K}\,(b/a)\,r}\,\textsf{Scope}$$

(b) Continuized CCG with side effects, fixing a monad $\langle \mathsf{M}, \eta, \star\rangle$ and a result type r.

Figure 1: Continuized CCGs with and without side effects, for a fixed result type r, monad $\langle \mathsf{M}, \eta, \star\rangle$, and empty string ε s.t. $\varepsilon \cdot \Gamma = \Gamma \cdot \varepsilon = \Gamma$. In both grammars, $\mathbf{S}\,m\,n := \lambda k.\,m\,(\lambda f.\,n\,(\lambda x.\,k\,(f\,x)))$.

tor M, an injection function η of type $a \to \mathsf{M}\,a$ (given any type a), and a recipe for sequencing programs $\star$, of type $\mathsf{M}\,a \to (a \to \mathsf{M}\,b) \to \mathsf{M}\,b$ (given any types a,b).[2] The monadic functors η and $\star$ are required to satisfy the following three *monad laws*, which ensure that $\eta\,x$ represents a *trivial* injection of x into the monad (left and right unit), and that order of evaluation, but not relative embedding, matters for program sequencing (a form of associativity):

Definition 1. For any monad $\langle \mathsf{M}, \eta, \star\rangle$:

Left unit:	$\eta\,x \star k = k\,x$
Right unit:	$m \star \eta = m$
Associativity:	$(m \star k) \star c = m \star \lambda x.\,k\,x \star c$

Monads provide an elegant bridge between side effects and continuations, because the type of $(\star)$ is equivalent to $\mathsf{M}\,a \to \mathsf{K}\,a\,(\mathsf{M}\,b)$. The sequencing function is thus itself a means of lifting effectful programs into continuized programs. Similarly, the injection function doubles as a default continuation, always available to be passed to $\mathsf{K}\,a\,(\mathsf{M}\,b)$ programs without generating any new side effects. In this way, we can construct continuized grammars directly from monads by trading in **Lift** for $\star$ and **Triv** for η (Figure 1b). Because **Scope** already operates at the level of continuized programs, it doesn't change.

This setup entails a form of *monadic functional application*, as in Fact 1. Moreover, **Lift** is a theorem of any monadic grammar, when we restrict the result type to $\mathsf{M}\,r$. See Fact 2.

Fact 1. $\mathbf{S}\,((\star)\,m)\,((\star)\,n)\,\eta$
$= m \star \lambda f.\,n \star \lambda x.\,\eta\,(f\,x)$

Fact 2. $\Gamma \vdash x : a \Rightarrow \Gamma \vdash \lambda k.\,k\,x : \mathsf{K}\,a\,(\mathsf{M}\,r)$

2 The arrow is a syntactically idle type constructor, s.t. $a \to b$ is inhabited by functions from type-a things to type-b things.

4 Finding the dynamic monad

We treat dynamic semantics as a case study for the monadic approach. Following Shan 2001, we characterize dynamic systems as those that recognize some forms of mutable state and nondeterminism. The former provides a mechanism by which expressions can manipulate the discourse context, usually through the introduction of discourse referents. The latter guarantees that indefinites are handled on a par with referential expressions, but whose referents are generated "randomly".

In classic formulations of dynamic semantics (e.g. Heim 1982, Kamp 1981, Groenendijk & Stokhof 1991, Dekker 1994), sentences denote relations on sequences. Non-empty relations correspond to truthful updates. Indefinites (and perhaps disjunctions) generate nonfunctional (in our terms, "nondeterministic") relations, with inputs matched against multiple outputs:

$$[\![\text{a linguist}]\!] = \lambda k i.\bigcup_{x \in \mathsf{ling}} k\,x\,(i + x)$$

$$[\![\text{a linguist left}]\!] = \lambda i.\,\{i + x : x \in \mathsf{ling} \wedge x \in \mathsf{left}\}$$

These denotations illustrate the way in which the standard dynamic approach puts state and nondeterminism front and center. Sentences are in fact *nothing but* nondeterministic modifications of state; even truth and falsity are derivative notions. Sentential conjunction amounts to relation composition, which pipes the sequences output by the left conjunct pointwise into the right.

Borrowing from the functional programming literature, we advocate a different perspective: treating nondeterminism and state modification as *side effects*, associated with semantic values, rather than replacing them. We pursue a monadic reformulation of dynamic semantics, by first locating a monad for state manipulation, and subsequently enriching it with nondeterminism.

Here, first, is the standard monad for state (Definition 2). Assume that γ is the type of "evaluation contexts". For our purposes, we set γ equal to the type of discourse referent ('dref') sequences.

Definition 2 (The State monad).

$$\begin{aligned} \mathsf{M}\,a &::= \gamma \to a * \gamma \\ \eta\, x &:= \lambda i.\langle x, i\rangle \\ m \star k &:= \lambda i.\, k\,(m\,i)_0\,(m\,i)_1 \end{aligned}$$

We model dref introduction as sequence extension (cf. de Groote 2006, Unger 2012, Charlow 2014) and dref retrieval as sequence projection. To dynamically charge a monadic individual (of type $\mathsf{M}\,e$), we pass the individual through a continuation that simply returns its underlying value and appends it to the output sequence (Definition 3). To take a simple example, $(\eta\,\mathsf{a})^{\triangleright} = \lambda i.\langle \mathsf{a}, i + \mathsf{a}\rangle$.

Keeping things simple, we assume a single projection function $\cdot_{\top}$ that returns as its value the most topical referent (say, for concreteness, the last) in a sequence (Definition 4).

Definition 3 (Monadic dref introduction).

$$m^{\triangleright} := m \star \lambda x i.\, \eta\, x\,(i + x)$$

Definition 4 (State monad dref retrieval).

$$\textbf{he} := \lambda i.\langle i_{\top}, i\rangle$$

As an example, consider the sentence *Al left* (call its denotation $\mathbf{X}$). Al is first injected into the state monad with η, then shifted so as to introduce himself to the discourse record, then sequenced into a continuation representing his nuclear scope. The result is a function from states to truth if Al left and falsity if he didn't, paired with an updated state which now lists Al as a dref:

$$(\eta\,\mathsf{a})^{\triangleright} \star \lambda x.\,\eta\,(\mathsf{left}\,x) = \lambda i.\langle \mathsf{left}\,\mathsf{a}, i + \mathsf{a}\rangle$$

Obversely, the sentence *he was tired* (call its denotation $\mathbf{Y}$) reads in a discourse state, chooses the most topical referent, and then returns the referent together with the original state, unmodified:

$$\textbf{he} \star \lambda x.\,\eta\,(\mathsf{tired}\,x) = \lambda i.\langle \mathsf{tired}\, i_{\top}, i\rangle$$

When we sequence the first sentence into the second (details anon), the dref introduced by *Al* is immediately picked up by the pronoun:

$$\begin{aligned} &\mathbf{X} \star \lambda p.\,\mathbf{Y} \star \lambda q.\,\eta\,(p \wedge q) \\ &\quad = \lambda i.\langle \mathsf{left}\,\mathsf{a} \wedge \mathsf{tired}\,\mathsf{a}, i + \mathsf{a}\rangle \end{aligned}$$

$$\begin{aligned} \textbf{john} &:= \mathsf{j} & \textbf{so} &:= \lambda i.\{\langle x, i\rangle \mid \mathsf{human}\, x\} \\ \textbf{left} &:= \mathsf{left} & \textbf{he} &:= \lambda i.\{\langle i_{\top}, i\rangle\} \end{aligned}$$

$$\begin{aligned} \textbf{not} &:= \lambda m i.\{\langle \neg\exists \pi \in m\,i.\,\pi_0, i\rangle\} \\ \textbf{a} &:= \lambda c i.\{\langle x, i'\rangle \mid \langle \mathsf{True}, i'\rangle \in c\,x\,i\} \\ \textbf{every} &:= \lambda c k.\,\textbf{not}\,(\textbf{a}\,c \star \lambda x.\,\textbf{not}\,(k\,x)) \\ \textbf{eo} &:= \lambda k.\,\textbf{not}\,(\textbf{so} \star \lambda x.\,\textbf{not}\,(k\,x)) \\ \textbf{nb} &:= \lambda k.\,\textbf{not}\,(\textbf{so} \star k) \end{aligned}$$

Figure 2: Lexicon specialized to State-Set Monad

The State monad treats discourse modification as a side effect of composition. To incorporate indefinites, which introduce drefs *indeterminately*, we need (similarly to standard dynamic analyses) to add a layer of nondeterminism to the basic State monad. This we do in Definition 5, which lays out what we will call the State-Set monad.[3]

Definition 5 (The State-Set monad).

$$\begin{aligned} \mathsf{M}\,a &::= \gamma \to (a * \gamma) \to t \\ \eta\, x &:= \lambda i.\{\langle x, i\rangle\} \\ m \star k &:= \lambda i.\bigcup_{\pi \in m\,i} k\,\pi_0\,\pi_1 \end{aligned}$$

Monadic meanings are now functions from discourse states to *sets* of values, tagged with potentially updated output states. Sequential combination still pipes the output state of one program in as input to another, but it now does so pointwise. In the next section, we will see how this provides a robust effects regime for modeling dynamic binding.

5 Examples

Figure 2 defines a small lexicon tailored for a CCCG built around the State-Set monad (e.g. **so** = ⟦someone⟧). We argue that this grammar offers a powerful vantage point for analyzing dynamic phenomena that interact with patterns of scope-taking.

These entries make no provisions for binding. Rather, dref introduction is treated modularly. We add a fourth and final combinator for generating a dref from a continuized expression (Definition 6). See Fact 3 for an indication of how continuized dref introduction relates to monadic dref introduction.

Definition 6 (Binding combinator).

$$\frac{}{\varepsilon \vdash \lambda m k.\, m\,(\lambda x i.\, k\,x\,(i + x)) : \mathsf{M}\,e / \mathsf{M}\,e}\,\blacktriangleright$$

Fact 3. $((\star)\,\textbf{so})^{\blacktriangleright} = \lambda k.\,\textbf{so}^{\triangleright} \star k$

3 Also known as the Parser monad (Hutton & Meijer 1998).

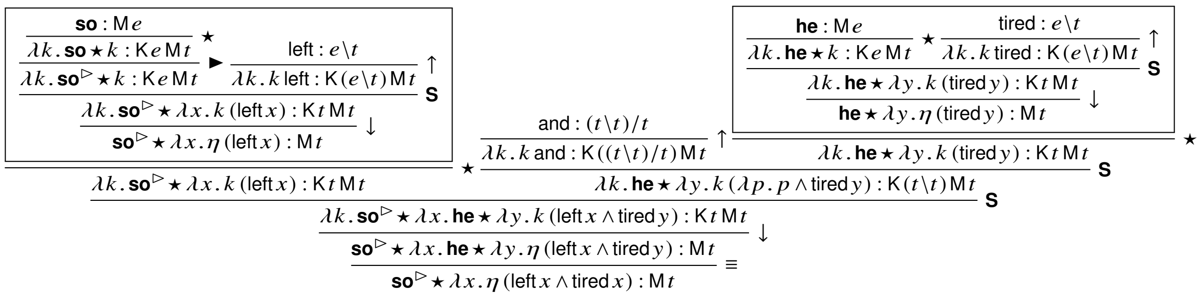

Figure 3: Deriving *someone$_i$ left; he$_i$ was tired.* Some abbreviatory conventions are exploited: $\uparrow$ indicates sequential application of η and $\star$ (i.e. type-lifting, cf. Fact 2), $\downarrow$ indicates that η has been introduced and consumed (i.e. lowering), and **S** stands in for both forward and backward continuized application (i.e. **Scope**). Explicit combinator introduction is likewise suppressed in favor of inference labels.

Figure 3 uses these tools to derive a case of cross-sentential anaphora. Notice that conjunction retains a static boolean semantics: since side effect composition is handled by the grammar, there is no reason to hard-code it into the semantics of sequencing. Relatedly, the predicates *left* and *tired* denote static $e\backslash t$ functions and interact with the monadic bits via η, $\star$, and **Scope**. A final property of the derivation worth highlighting is that both clauses are *evaluated*, i.e. applied to the default continuation η, before they are conjoined (the $\boxed{\text{boxes}}$ are simply intended to highlight this fact). This delimits the scope-taking of any expressions in the individual conjuncts, including that of the $\star$'d indefinite in the first sentence. In a manner of speaking, the indefinite hasn't been allowed to "QR" out of its clause. Nevertheless, its side effect — the introduction of a nondeterministic dref — lives on to influence the evaluation of the pronoun in the second conjunct.

Other operators, however, may *quantify* over the nondeterminism of indefinites in their scope, gobbling up their side effects in the process. Negation is the classic example. As defined in Figure 2, **not** evaluates its prejacent, discards any discourse referents generated in the process, and then checks that none of the alternative updates resulted in true boolean values. As is standard, we can use this state-capturing behavior to build up other dynamically closed meanings. For example, the universal quantifier **eo** (= ⟦everyone⟧) defined in Figure 2 will display quite different behavior than the indefinite of Figure 3 when evaluated at the clause boundary. Plugging in the unit continuation will do two things: (i) as before, it will close off the nuclear scope of the universal; but (ii) this time, we will be left with a pure (effect-free) computation, equivalent to $\eta\,(\forall x.\,\mathsf{human}\,x \Rightarrow \mathsf{left}\,x)$. The same goes, mutatis mutandis, for **nb** (= ⟦nobody⟧).

Importantly, this does not preclude the possibility of donkey anaphora (e.g. the classic *every farmer who owns a donkey$_i$ beats it$_i$*). Returning to the lexical entry for **every**, note that the restrictor c here acquires a kind of monadic scope, via $\star$, over the nuclear scope k. This means that any side effects inside c will influence the context of evaluation for k. However, as in other dynamic systems, the wide-scoping negation will discharge whatever side effects are accrued in the process.

6 Discussion

Monads thus offer a natural way to extend CCGs with tools for dynamic binding. The combinatory apparatus we have proposed has four essential pieces: η, $\star$, $\blacktriangleright$, and **Scope**. Along with some intuitive lexical entries, this allows indefinites to extend their binding scope beyond the domain in which they are evaluated — without recourse to dynamic conjunction, and within a completely standard model theory where "contexts of evaluation" are constructed on the fly (cf. van Eijck 2001, de Groote 2006).

The theory can be scaled up in the usual ways to offer a robust characterization of a wide range of exceptional binding configurations, including dynamic binding by quantifiers such as *exactly one linguist* (Charlow 2014). Moreover, though we lack the space to consider this here, the means by which indefinites extend their binding scope turns out to offer an immediate explanation for their ability to take "quantificational" scope out of scope islands (ibid.), and underlies an attractive dynamic account of functional quantification and pair-list phenomena (Bumford to appear).

As we indicated at the outset, the basic strategy at play here is quite general. Indeed, *any* sufficiently "well-behaved" (see Definition 1) regime of semantic enrichment (e.g. focus-sensitivity, alternative-generation, intensionality) may be accommodated along these lines. Furthermore, because of the inherent modularity of the monadic approach, various side effects may be added to or subtracted from the grammar without adjusting the basic compositional machinery or the lexical entries that are not sensitive to the effects in question. Therefore, we expect these results will be of interest for natural language semanticists of many stripes, in addition to categorial grammarians working on donkey anaphora and/or scope-taking.

References

Barker, Chris & Chung-chieh Shan. 2008. Donkey Anaphora is In-Scope Binding. *Semantics & Pragmatics* 1(1). 1–46. http://dx.doi.org/10.3765/sp.1.1.

Barker, Chris & Chung-chieh Shan. 2014. *Continuations and Natural Language*. Oxford: Oxford University Press.

Barwise, Jon & Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy* 4(2). 159–219. http://dx.doi.org/10.1007/BF00350139.

Bumford, Dylan. to appear. Incremental quantication and the dynamics of pair-list phenomena. *Semantics & Pragmatics* 8(9).

Charlow, Simon. 2010. Two kinds of binding out of DP. Unpublished ms.

Charlow, Simon. 2014. *On the semantics of exceptional scope*: New York University Ph.D. thesis.

Dekker, Paul. 1994. Predicate Logic with Anaphora. In Mandy Harvey & Lynn Santelmann (eds.), *Proceedings of Semantics and Linguistic Theory 4*, 79–95. Ithaca, NY: Cornell University.

van Eijck, Jan. 2001. Incremental Dynamics. *Journal of Logic, Language and Information* 10(3). 319–351. http://dx.doi.org/10.1023/A:1011251627260.

Groenendijk, Jeroen & Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39–100. http://dx.doi.org/10.1007/BF00628304.

de Groote, Philippe. 2006. Towards a Montagovian account of dynamics. In Masayuki Gibson & Jonathan Howell (eds.), *Proceedings of Semantics and Linguistic Theory 16*, 1–16. Ithaca, NY: Cornell University.

Heim, Irene. 1982. *The Semantics of Definite and Indefinite Noun Phrases*: University of Massachusetts, Amherst Ph.D. thesis.

Hutton, Graham & Erik Meijer. 1998. Monadic parsing in Haskell. *Journal of Functional Programming* 8. 437–444. http://dx.doi.org/10.1017/S0956796898003050.

Kamp, Hans. 1981. A theory of truth and semantic interpretation. In Jeroen Groenendijk, Theo Janssen & Martin Stokhof (eds.), *Formal Methods in the Study of Language*, 277–322. Mathematical Centre Amsterdam.

Kamp, Hans & Uwe Reyle. 1993. *From Discourse to Logic*. Dordrecht: Kluwer Academic Publishers.

Liang, Sheng, Paul Hudak & Mark Jones. 1995. Monad Transformers and Modular Interpreters. In *22nd ACM Symposium on Principles of Programming Languages (POPL '95)*, 333–343. ACM Press.

Moggi, Eugenio. 1989. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in computer science*, 14–23. Piscataway, NJ, USA: IEEE Press.

Montague, Richard. 1974. The proper treatment of quantification in ordinary English. In Richmond Thomason (ed.), *Formal Philosophy*, chap. 8, 247–270. New Haven: Yale University Press.

Shan, Chung-chieh. 2001. A Variable-Free Dynamic Semantics. In Robert van Rooy & Martin Stokhof (eds.), *Proceedings of the Thirteenth Amsterdam Colloquium*, University of Amsterdam.

Shan, Chung-chieh. 2002. Monads for natural language semantics. In Kristina Striegnitz (ed.), *Proceedings of the ESSLLI 2001 Student Session*, 285–298.

Shan, Chung-chieh. 2005. *Linguistic Side Effects*: Harvard University Ph.D. thesis.

Shan, Chung-chieh & Chris Barker. 2006. Explaining Crossover and Superiority as Left-to-right Evaluation. *Linguistics and Philosophy* 29(1). 91–134. http://dx.doi.org/10.1007/s10988-005-6580-7.

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

Wadler, Philip. 1992. Comprehending monads. In *Mathematical Structures in Computer Science*, vol. 2 (special issue of selected papers from 6th Conference on Lisp and Functional Programming), 461–493.

Wadler, Philip. 1994. Monads and composable continuations. *Lisp and Symbolic Computation* 7(1). 39–56. http://dx.doi.org/10.1007/BF01019944.

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