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 enrich a programming language or natural language grammar with *side effects* (cf. Shan 2005). 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 with potential side effects. Our proposal is to replace the standard lifting and lowering operations of CCCGs, with $\star$ and η, respectively. Combined with some intuitive lexical entries, this has the automatic effect of producing a CCCG that countenances side effects. We show that any system of effects can be grafted onto a continuized CCG in this way, and illustrate with a single in-depth case study, the case of dynamic binding. Crucially, given the underlying continuation-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.

Again, any sufficiently "well-behaved" (see Section 5) regime of semantic enrichment (e.g. focus-sensitivity, alternative-generation, intensionality) may be accommodated along similar lines. Furthermore, because of the inherent modularity of the monadic approach, various side effects may be added or removed to 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.

2 Continuations

Simplifying somewhat, we take it that a standard CCCG consists of three overloaded 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. The inference rules make use of a polymorphic type constructor K that is parameterized by two concrete types. 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. Formally, $\mathsf{K}\,a\,r \equiv (a \to r) \to r$, where $\alpha \to \beta$ types a (direction-insensitive) function from type α to type β. The $(a \to r)$ argument to something of type $\mathsf{K}\,a\,r$ is called its *continuation*, and represents the denotation of its context.

lift turns any expression into a function of its *continuation*. It is a polymorphic generalization of the familiar lifting procedure that converts individuals to GQs. At the heart of a CCCG, **scope** defines the compositional process whereby two continuized expressions are combined, giving the left scope over the right. Finally, **triv** is a trivial continuation used to delimit contexts of evaluation and expose underlying semantic values. Shan & Barker 2006 show that these three operations suffice to derive inverse scope, and as a corollary, backward function composition.

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

$$\frac{}{\varepsilon \vdash \lambda x.\, x : a \to a}\,\textbf{triv} \qquad \frac{}{\varepsilon \vdash \lambda x k.\, k\, x : a/\mathsf{K} a\, r}\,\textbf{lift} \qquad \frac{}{\varepsilon \vdash \mathbf{S} : (\mathsf{K} b r/\mathsf{K} a r)/\mathsf{K}(b/a) r}\,\textbf{scope}$$

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

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

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

(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 and monad $\langle \mathsf{M}, \eta, \star \rangle$. In both grammars, $\mathbf{S} \equiv \lambda m m' k\, .\, m\, (\lambda f\, .\, m'\, (\lambda x\, .\, k\, (f\, x)))$.

3 Adding side effects

One nice property of continuized grammars is that they do not place any restrictions on the type of the return value, i.e., the type of the context. For simple cases where only scope-taking is at-issue, we can use type t, the type of sentences, over which quantificational operators typically scope. But in general, we may be interested in contexts that denote in richer semantic spaces, as is typically the case with questions, focus-bearing constituents, 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 constructor 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). 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 (Monad laws). 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 bind function is thus itself a means of lifting effectful programs into continuized programs. Similarly, the unit function doubles as a default continuation, always available to be passed in 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. Conveniently, because **scope** operates at the level of already continuized programs, it doesn't change.

In fact, **lift** is a theorem of any monadic grammar, when we restrict the result type to $\mathsf{M} r$. Generally, for any monad $\langle \mathsf{M}, \eta, \star \rangle$ and result type r:

Fact 1. $\Gamma \vdash x : a \Rightarrow \Gamma \vdash \lambda k.\, k\, x : \mathsf{K} a\, (\mathsf{M} r)$

Fact 2. $\mathbf{S}\, ((\star)\, m)\, ((\star)\, n)\, \eta = m \star \lambda f.\, n \star \lambda x.\, \eta\, (f\, x)$

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".[1]

In classic formulations of dynamic semantics (Heim 1982, Kamp 1981, Groenendijk & Stokhof

1 Dynamic treatments following Groenendijk & Stokhof 1990 (e.g. Zimmermann 1991, Dekker 1993, Szabolcsi 2003, de Groote 2006) provide a way for indefinites to extend their binding domain but do not treat indefinites as nondeterministic analogs of proper names.

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. Sentential conjunction amounts to relation composition, which pipes the sequences output by the left conjunct pointwise into the right.

(1)

$$[\![\text{someone}]\!] = \lambda k i. \bigcup_{x \in \text{person}} k\, x\, (i + x)$$

$$[\![\text{someone left}]\!] = \lambda i. \{i + x : x \in \text{person} \wedge x \in \text{left}\}$$

The denotations in (1) illustrate the way in which the standard dynamic approach puts state and nondeterminism are put front and center. Sentences are in fact *nothing but* nondeterministic modifications of state; even truth and falsity are derivative notions. 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.

Here is the standard monad for state manipulation. Assume that γ is the type of "evaluation contexts". For our purposes, we set γ equal to the type of discourse referent sequences.

Definition 2 (The State monad).

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

To keep things simple, we model dref introduction as sequence extension (cf. de Groote 2006, Unger 2012, Charlow 2014), and dref retrieval as sequence projection. Again, for simplicity we assume a single projection function $\cdot_\top$ that picks out the most topical referent (for concreteness, the last).

Definition 3 (Dref introduction).

$$m^{\triangleright} := m \star \lambda x i. \langle x, i + x \rangle$$

Definition 4 (Dref retrieval).

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

An example, *Al left* (call this **X**):

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

Pronoun sentence *he was tired* (call this **Y**):

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

Sequencing the two. The dref introduced by the proper name in the first sentence is accessed by the pronoun in the second.

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

So we have state modification as a side effect. To say something about indefinites, i.e. to allow them to refer and introduce drefs nondeterministically, we need to enrich the state monad with nondeterministic side effects. The monad for nondeterminism is the Set monad, given in Definition 5:

Definition 5 (The Set monad).

$$\begin{aligned} \mathsf{M}\, a &::= a \to t \\ \eta\, x &:= \{x\} \\ m \star k &:= \bigcup_{x \in m} k\, x \end{aligned}$$

Use StateT to stitch the two together. Given any monad $\mathcal{M} = \langle \mathsf{L}, \eta_\mathsf{L}, \star_\mathsf{L} \rangle$, StateT is a recipe for building a new monad with which adds State-type functionality to $\mathcal{M}$:[2]. Result also known as the Parser monad. Hutton & Meijer 1998

Definition 6 (The StateT monad transformer).

$$\begin{aligned} \mathsf{M}\, a &::= \gamma \to \mathsf{L}\, (a \times \gamma) \\ \eta\, x &:= \lambda i. \eta_\mathsf{L} \langle x, i \rangle \\ m \star k &:= \lambda i. m\, i \star_\mathsf{L} \lambda \pi. k\, \pi_0\, \pi_1 \end{aligned}$$

Definition 7 (The State_Set monad).

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

Static lexicon, dynamic lexicon
Modular treatment of binding.

Previous : $\mathbf{bind}\, m := \lambda k. m\, (\lambda x. k\, x\, x)$
Proposal : $\mathbf{bind}\, m := \lambda k. m\, (\lambda x i. k\, x\, (i + x))$

5 Examples

Cross-sentential anaphora. Let us assume that *scope islands*, e.g., tensed clauses, need to be evaluated—i.e., lowered (cf. Barker 2002, Barker & Shan 2008). In practice, this means a tensed clause must pass through a stage where it denotes something of type

2 Fn. about SetT

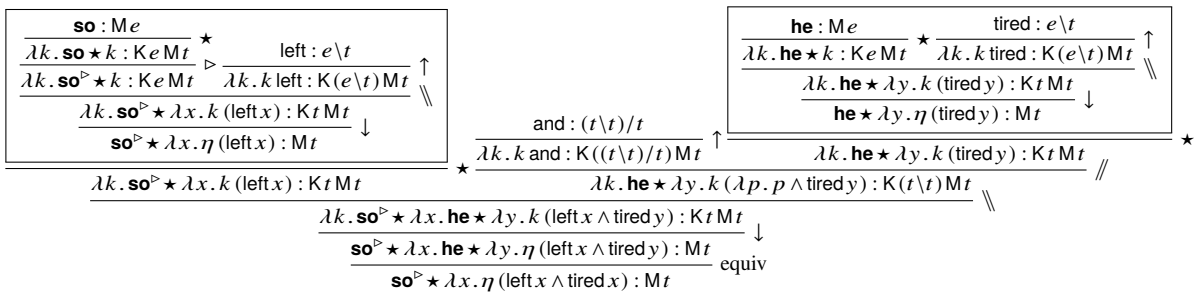

Figure 2: Cross-sentential anaphora: deriving *someone$_i$ left; he$_i$ was tired.*

Mt. There are a variety of options for enforcing this syntactically, but here we concentrate on the semantic upshots of forced evaluation.[3] the indefinite's side effects influence the evaluation of the second clause, even as the indefinite scopes within its clause. See Figure 2.

Negation. Requires there to be no true boolean value returned, tosses out any discourse referents generated in its scope. (Standard). Use to define dynamically closed meanings (e.g. conditional, universal quantifier, etc.)

$$\begin{aligned}\textbf{not} &:= \lambda ms.\{\langle \neg\exists \pi \in m\, s.\, \pi_0, s\rangle\}\\ \textbf{no.ling} &:= \lambda k.\, \textbf{not}\,(\textbf{a.ling} \star k)\\ \textbf{ev.ling} &:= \lambda k.\, \textbf{not}\,(\textbf{a.ling} \star \lambda x.\, \textbf{not}\,(k\, x))\end{aligned}$$

Compare universals. After ending the derivation at the clause boundary, we're left with a pure computation. The universal's side effects have died on evaluation.

$$\eta\,(\forall x.\, \mathsf{ling}\, x \Rightarrow \mathsf{left}\, x)$$

Donkey anaphora works similarly. Take the following. The restrictor c here acquires a kind of monadic scope, via $\star$, over the nuclear scope k. This means any side effects inside c influence the context of evaluation for k. However, once k is grabbed, the wide-scoping negation discharges side effects (as is standard in dynamic systems).

$$\textbf{every} := \lambda ck.\, \textbf{not}\,(\textbf{a}\, c \star \lambda x.\, \textbf{not}\,(k\, x))$$

6 Discussion

Compare PLA, where only sentences are imbued with context change potential. Necessary since in PLA and standard dynamic treatments of anaphora, discourse-level content and truth-conditional content are conflated—i.e. a sentence denotes a non-empty relation on sequences iff the sentence is true. Thus: standard dynamic techniques (DPL, DMG) not reducible to monads.

Some upshots: no dynamic conjunction, completely standard model theory (cf. de Groote 2006). "Contexts of evaluation" are constructed on the fly. Variable-free, directly compositional (Jacobson 1999).

Monads as a natural way to extend a continuations-based grammar with tools for dynamic binding and exceptional scope. In the end: you have functional application, plus the functors from whichever monads are implicated in a given language. Effects recognized in the types.

There is no need to settle on a single ("the") grammar. Different and quite varied side effects regimes can be modularly grafted onto a simple applicative ("pure") core. Lexical entries that would seem incongruous in a flat-footed standard perspective integrate seamlessly in a single grammar.

Theory extends to scope islands, wide range of exceptional binding configurations Charlow 2014. Extends to pair-list phenomena, functional quantification: Bumford to appear. Crossover, superiority less clear (cf. Shan & Barker 2006, Barker & Shan 2008).

Broader question: how this relates to the idea that continuations can simulate any monad (Filinski 1994). I don't understand this result well enough to say anything (Dylan?).

3 In terms of LF, forcing evaluation of a scope island corresponds to disallowing QR out of the scope island.

References

Barker, Chris. 2002. Continuations and the Nature of Quantification. *Natural Language Semantics* 10(3). 211–242. http://dx.doi.org/10.1023/A:1022183511876.

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.

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. 1993. *Transsentential meditations: ups and downs in dynamic semantics*: University of Amsterdam 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.

Filinski, Andrzej. 1994. Representing Monads. In *Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 446–457. New York: ACM Press.

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

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.

Jacobson, Pauline. 1999. Towards a Variable-Free Semantics. *Linguistics and Philosophy* 22. 117–184. http://dx.doi.org/10.1023/A:1005464228727.

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.

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.

Szabolcsi, Anna. 2003. Binding on the Fly: Cross-Sentential Anaphora in Variable-Free Semantics. In Geert-Jan M. Kruijff & Richard T. Oehrle (eds.), *Resource-Sensitivity, Binding and Anaphora*, 215–227. Dordrecht: Kluwer Academic Publishers.

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.

Zimmermann, Thomas Ede. 1991. Dynamic logic and case quantification. In Martin Stokhof, Jeroen Groenendijk & David Beaver (eds.), *Quantification and Anaphora I* (DYANA Deliverable R2.2.A), 191–195.