A CONTINUATION SEMANTICS OF INTERROGATIVES THAT ACCOUNTS FOR BAKER'S AMBIGUITY

CHUNG-CHIEH SHAN

Abstract. Wh-phrases in English have two important properties: first, they can appear both raised and in-situ; second, while in-situ wh-phrases can take semantic scope beyond the immediately enclosing clause, raised ones cannot. I present a denotational semantics of interrogatives that naturally accounts for these two properties; it neither conducts movement at a level of semantic representation separate from surface syntax nor posits lexical ambiguity between raised and in-situ occurrences of the same wh-phrase. My analysis is based on the concept of *continuations*; in particular, it uses a novel type system for higher-order continuations to handle wide-scope wh-phrases while remaining within a Montague-style framework of grammar. This treatment sheds light on the combinatorics of interrogatives as well as other kinds of so-called A′-movement.

1. Introduction

Baker (1968) studied multiple-wh questions such as those in (1).

(1) a. Who remembers where we bought what?
b. Who do you think remembers what we bought for whom?

Each question in (1) contains three wh-phrases and is ambiguous between two readings with different notions of what constitutes an appropriate answer.

(2) Who remembers where we bought what?
a. John remembers where we bought the vase.
b. John remembers where we bought what.

(3) Who do you think remembers what we bought for whom?
a. I think John remembers what we bought for Mary.
b. I think John remembers what we bought for whom.

Intuitively, both cases of ambiguity are because the final wh-phrase—*what* in (1a) and *whom* in (1b)—can take either wide scope (2a, 3a) or narrow scope (2b, 3b).[1]

In this paper, I will focus on two properties of interrogatives:

- Wh-phrases appear both raised and in-situ. For example, in (1b), *who* and *what* appear raised while *whom* appears in-situ.

Date: October 22, 2001 (08:27:45; revision 1.3).

This paper started out as a term paper for MIT 24.979 in Spring 2001 (Kai von Fintel and Irene Heim, Topics in Semantics: Questions and Focus). Thanks to the participants of the class, Chris Barker, Pauline Jacobson, Stuart Shieber, and Dylan Thurston for discussions and comments. This work is supported by the National Science Foundation under Grant IRI-9712068.

[1]For now, I will disregard the distinction between questions that allow or expect *pair-list* answers (e.g., *John remembers where we bought the vase, and Sue remembers where we bought the table*) and questions that require or expect non-pair-list answers.

- Raised wh-phrases must take semantic scope exactly over the clause they are raised to overtly. For example, in (1b), *who* must take wide scope, and *what* must take narrow scope. Only *whom* has ambiguous scope; accordingly, the question has only 2 readings, not 4 or 8.

I will present a strictly compositional semantics of interrogatives in English that accounts for these properties. Specifically, in my analysis,

- there is no covert movement or wh-raising between surface syntax and denotational semantics ("at LF"; "after Spell-Out"; "Quantifying-In"), yet a single denotation suffices for both raised and in-situ appearances of each wh-phrase. Moreover,
- as a natural consequence of the denotation of wh-phrases and the type-shift operations available in the grammar, only in-situ appearances of wh-phrases can have ambiguous semantic scope.

I will describe my system as one where, roughly speaking, interrogative clauses denote functions from answers to propositions.[2] However, as I will discuss in §4.4, such denotations are not essential for my purposes—the essential ideas in my analysis carry over easily to a system where interrogative clauses denote say sets of propositions instead. Therefore, this paper bears not so much on what interrogatives denote, but how.

My analysis builds upon Barker's use of *continuations* to characterize quantification in natural language (2000a, 2000b). In §2 below, I will review the concept of continuations and how Barker assigned to quantificational NPs such as *everyone* denotations that manipulate continuations. The system I will present generalizes Barker's semantics in several aspects, which I will point out as we encounter. In §3, I will extend Barker's semantics with interrogative denotations, which manipulate the *answer type* of continuations. I will then explain how this treatment of interrogatives accounts for the properties above. In doing so, I will not be concerned with the semantics of verbs such as *know* that take interrogative complements, but rather with how to derive denotations for interrogative clauses themselves. Finally, in §4, I will conclude with some speculations on further applications of my treatment, for example to explain superiority effects.

The presentation in this paper will loosely follow the framework of categorial grammar, in particular Combinatory Categorial Grammar (Steedman 1987, 1996). However, I expect the central insights to be easy to adapt to other frameworks of compositional semantics.

2. Continuation semantics

Continuations are a well-known and widely applied concept in programming language semantics. In brief, "the continuation represents an entire (default) future for the computation" (Kelsey, Clinger, Rees, et al. 1998). Danvy (2001) said that continuations "can be explained in five minutes, but grasping them seems to require a lifetime." What follows is a brief introduction to continuations in first programming language and then natural language terms.

2.1. **Programming language continuations.**

Consider the following expression in a hypothetical programming language.

(4) **if** $2 \times 3 = 6$ **then** "equal" **else** "unequal"

[2] Hirschbühler (1978) cited a variety of such proposals and argued against those of Hausser (1978) and Hausser and Zaefferer (1979) in particular. In this paper, I will not directly address Hirschbühler's concerns.

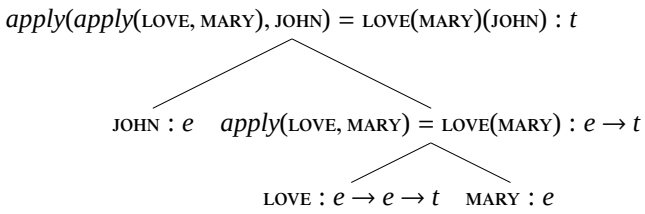

FIGURE 1. *John loves Mary*

To evaluate the expression (4), we may begin by evaluating the subexpression 2×3. Once we know the numeric value of 2×3, we can then compare it to 6 and return either the string "equal" or the string "unequal" depending on the result of the comparison. With respect to the subexpression 2×3, then, the *context* of evaluation is the "expression with a hole"

(5) **if** ____ = 6 **then** "equal" **else** "unequal".

This evaluation context is essentially a map

(6) $c = \lambda x.$ **if** $x = 6$ **then** "equal" **else** "unequal"

from numbers to strings—in particular, $c(6)$ = "equal". The map c is called the *continuation* of the subexpression 2×3 in the expression (4), and is assigned the type

$$number \rightsquigarrow string,$$

where *number* and *string* are base types, and $\rightsquigarrow$ is a binary type constructor. The *value type* of a continuation is its domain, and the *answer type* of a continuation is its codomain; for example, the continuation c has value type *number* and answer type *string*.[3] I distinguish between the continuation type $number \rightsquigarrow string$ and the function type $number \to string$, even though they may have the same models and I use the same λ notation for both kinds of abstractions. The reason for this distinction will become clear in §3.

With the continuation c in hand, we can call $c(2 \times 3)$ to check whether 2×3 is equal to 6, as well as call $c(2 + 3)$ to check whether $2 + 3$ is equal to 6. We can play "what-if" with the hole in (5), plugging in different values to see what the top-level answer would come out to be.

2.2. **Natural language continuations.** The concepts of context, continuation, value type and answer type transfer directly from programming language semantics to Montague grammar for natural language. In basic Montague grammar, the meaning of *John loves Mary* is computed compositionally from denotations assigned to the lexical entries *John*, *loves*, and *Mary*. Starting with the denotations

$$\text{JOHN} : e, \qquad \text{LOVE} : e \to e \to t, \qquad \text{MARY} : e,$$

where e is the base type of individuals and t is the base type of propositions, we recursively apply the composition rule

(7)
$$\begin{array}{c} apply(x, y) = x(y) : \beta \\ \hline x : \alpha \to \beta \quad y : \alpha \end{array}$$

[3]The concept of answer types in continuation semantics is not (directly) related to the concept of appropriate answerhood in interrogatives.

to obtain a top-level denotation of type t, as shown in Figure 1. (Throughout this paper, I use the Greek letters α, β, γ, and δ to represent type variables, in other words variables that can be instantiated with any type.[4] Also, by convention, all binary type constructors associate to the right. For example, the type $e \to e \to t$, unparenthesized, means $e \to (e \to t)$.) In the expression *John loves Mary*, the continuation of the subexpression *Mary* is

$$c = \lambda x.\, \textsc{love}(x)(\textsc{john}) : e \rightsquigarrow t,$$

which corresponds to the context

(8) John loves ____.

With this continuation in hand, we can call $c(\textsc{mary})$ to check whether John loves Mary, as well as call

$$\forall x.\, \textsc{person}(x) \Rightarrow c(x) \tag{9}$$

to check whether John loves everyone, playing "what-if" with the hole in (8). This is the intuition behind the continuation semantics treatment of quantification.

2.3. **Continuizing grammars.** Formally, given any Montague-style grammar, we can *continuize* it to obtain another Montague-style grammar by following the following four steps.[5] First, introduce a new "lift" rule

$$\begin{array}{c} \ell(x) = \lambda c.\, c(x) : (\alpha \rightsquigarrow \gamma) \to \gamma \\ | \\ x : \alpha \end{array} \quad . \tag{10}$$

The type-shift operator ℓ defined here is just the lifting operation familiar from Montague's treatment of quantification (1974), except the continuation type $\alpha \rightsquigarrow \gamma$ is distinct from the function type $\alpha \to \gamma$. The type $(\alpha \rightsquigarrow \gamma) \to \gamma$ can be thought of the type of a *continuized* α-value; each continuized value takes a continuation as input and returns a final answer.

Second, introduce a new "evaluate" rule

$$\begin{array}{c} \varepsilon(q) = q(\lambda x.\, x) : \gamma \\ | \\ q : (\alpha \rightsquigarrow \alpha) \to \gamma \end{array} \quad . \tag{11}$$

The type-shift operator ε takes as input a continuized α-value q, and feeds to it the identity continuation $\lambda x.\, x$. It is easy to verify that $\varepsilon(\ell(x)) = x$ for all x; in other words, ℓ is an injection and ε is its inverse.

Third, for each existing unary rule, say

$$\begin{array}{c} f(x) : \tau' \\ | \\ x : \tau \end{array}$$

where f is some operator mapping a type τ to another type τ', add a new unary rule

$$\begin{array}{c} f^*(q) = \lambda c.\, q(\lambda x.\, c(f(x))) : (\tau' \rightsquigarrow \gamma_1) \to \gamma_2 \\ | \\ q : (\tau \rightsquigarrow \gamma_1) \to \gamma_2 \end{array} \quad . \tag{12}$$

[4] In other words, what this paper calls types are known as *type-schemes* in the Hindley-Milner type system (Hindley and Seldin 1986).

[5] To simplify the presentation, I assume that all productions in the grammar are either unary-branching or binary-branching. Also, the technical machinery I use here to continuize a grammar is different from that used by Barker. For instance, Barker did not use dynamic type-shifting, but I do here (10, 11).

Intuitively, this step upgrades any existing type-shift operator f in the grammar that operates on τ-values x to a new type-shift operator f^* that operates on continuized τ-values q.

Fourth and finally, for each existing binary rule, say

$$\begin{array}{c} g(x,y) : \tau' \\ \diagup \quad \diagdown \\ x : \tau_1 \quad y : \tau_2 \end{array}$$

where g is some composition method mapping some types τ_1 and τ_2 to another type τ', add two new binary rules[6]

(13a)
$$\begin{array}{c} g^{>}(q,r) = \lambda c.\, q(\lambda x.\, r(\lambda y.\, c(g(x,y)))) : (\tau' \rightsquigarrow \gamma_1) \to \gamma_3 \\ \diagup \quad \diagdown \\ q : (\tau_1 \rightsquigarrow \gamma_2) \to \gamma_3 \quad r : (\tau_2 \rightsquigarrow \gamma_1) \to \gamma_2 \end{array} ,$$

(13b)
$$\begin{array}{c} g^{<}(q,r) = \lambda c.\, r(\lambda y.\, q(\lambda x.\, c(g(x,y)))) : (\tau' \rightsquigarrow \gamma_1) \to \gamma_3 \\ \diagup \quad \diagdown \\ q : (\tau_1 \rightsquigarrow \gamma_1) \to \gamma_2 \quad r : (\tau_2 \rightsquigarrow \gamma_2) \to \gamma_3 \end{array} .$$

Intuitively, this step upgrades any existing composition method g in the grammar that operates on τ_1- and τ_2-values x and y to two new composition methods $g^{>}$ and $g^{<}$ that operate on continuized-τ_1- and continuized-τ_2-values q and r. In programming language terms, $g^{>}$ (13a) evaluates the left branch before the right branch, whereas $g^{<}$ (13b) evaluates the right branch before the left branch.

For instance, given the composition rule (7) for direct function application, we can set $g = \mathit{apply}$, $\tau_1 = \alpha \to \beta$, $\tau_2 = \alpha$, and $\tau' = \beta$ in (13) to obtain the upgraded composition rules

(14a)
$$\begin{array}{c} \mathit{apply}^{>}(q,r) = \lambda c.\, q(\lambda x.\, r(\lambda y.\, c(x(y)))) : (\beta \rightsquigarrow \gamma_1) \to \gamma_3 \\ \diagup \quad \diagdown \\ q : ((\alpha \to \beta) \rightsquigarrow \gamma_2) \to \gamma_3 \quad r : (\alpha \rightsquigarrow \gamma_1) \to \gamma_2 \end{array} ,$$

(14b)
$$\begin{array}{c} \mathit{apply}^{<}(q,r) = \lambda c.\, r(\lambda y.\, q(\lambda x.\, c(x(y)))) : (\beta \rightsquigarrow \gamma_1) \to \gamma_3 \\ \diagup \quad \diagdown \\ q : ((\alpha \to \beta) \rightsquigarrow \gamma_1) \to \gamma_2 \quad r : (\alpha \rightsquigarrow \gamma_2) \to \gamma_3 \end{array} .$$

The first composition method $\mathit{apply}^{>}$ evaluates q into a function x before evaluating r into an argument y and applying x to y. The second composition method $\mathit{apply}^{<}$ evaluates r into an argument y, then evaluates q into a function x and applies x to y.

Starting with "pure" Montague grammar—where the only mode of composition is function application, and there are no type-shift operations—we can continuize it into a new grammar by following the four steps above. The new grammar has two type-shift operations, namely ℓ (10) and ε (11), and three composition methods, namely direct function application (7) and its two upgraded cousins (14). This is illustrated in Figure 2.

2.4. **Quantification.** I am now ready to demonstrate how continuation semantics treats quantification, for example in the sentences

(15) a. John loves everyone.
b. Everyone loves Mary.

[6]The types here clearly correspond to the inference rules of Danvy and Filinski (1989, §5), and are the most general types that can be assigned under the Hindley-Milner type system (Hindley and Seldin 1986).

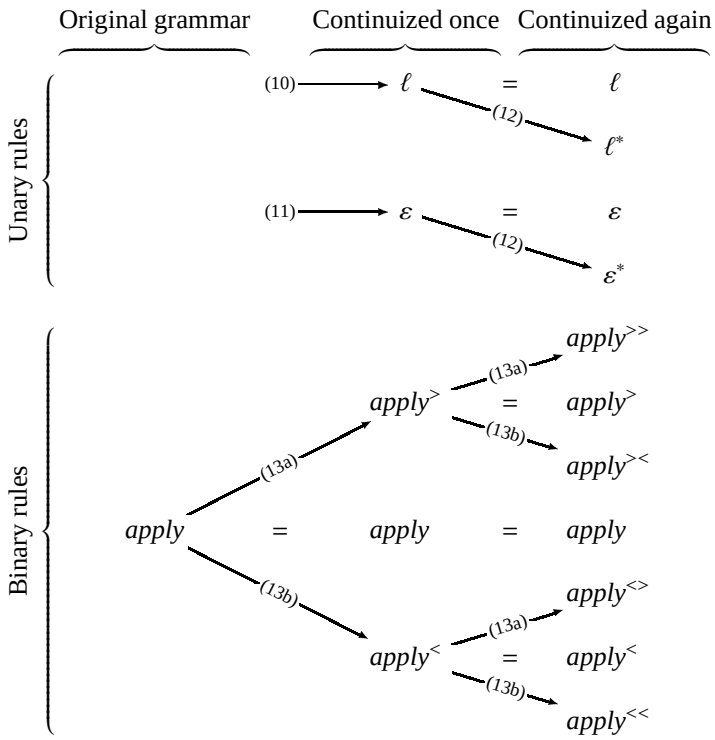

FIGURE 2. Continuizing Montague grammar. Starting from only direct function application, continuizing once gives a grammar with 3 binary rules and 2 unary rules. Continuizing again gives a grammar with 7 binary rules and 4 unary rules.

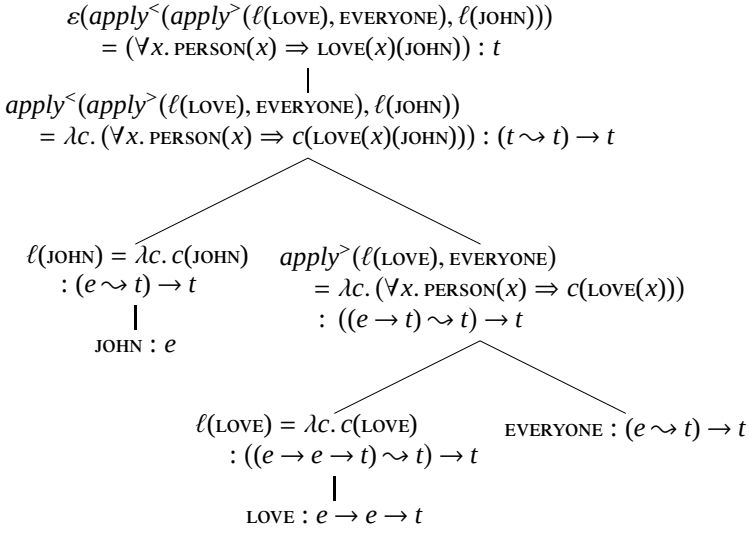

FIGURE 3. *John loves everyone*

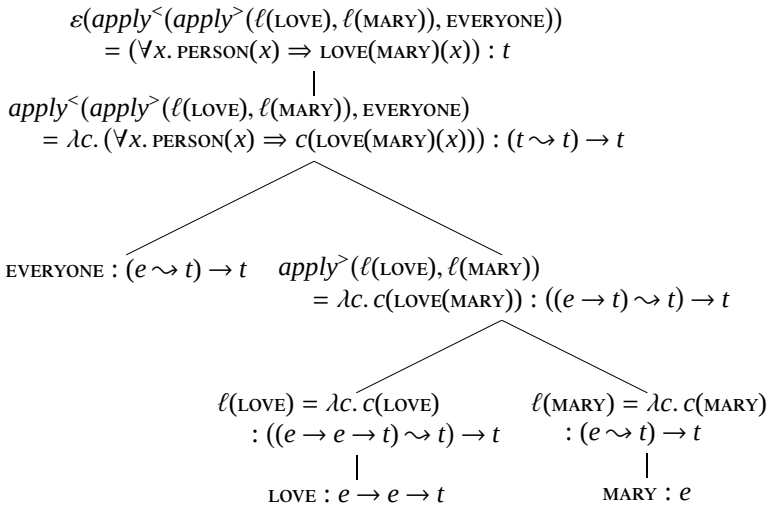

FIGURE 4. *Everyone loves Mary*

Following the intuition from §2.2 that a continuation is an expression with a hole, let us assign to *everyone* the denotation

(16) $$\text{EVERYONE} = \lambda c.\, (\forall x.\, \text{PERSON}(x) \Rightarrow c(x)) : (e \rightsquigarrow t) \rightarrow t.$$

(Note that the type $(e \rightsquigarrow t) \rightarrow t$ is that of a continuized e-value and familiar from Montague's treatment of quantification.) We can now derive denotations for (15a) and (15b), as shown in Figures 3 and 4.

The derivations for (15a) and (15b) shown here are not the only ones possible. For example, where we used the composition method $apply^{>}$ (14a), we could have used the composition method $apply^{<}$ (14b) instead, and vice versa. For the two sentences considered here, all derivations that end at the type t give semantically equivalent results. However, for other sentences, such as *someone loves everyone*, different derivations—more precisely, different orders of evaluation—give differently scoped results.

Space constraints prevent me from elaborating on Barker's analysis of quantification; I refer the reader to his papers for linguistic motivation and additional details. Meanwhile, it is worth pointing out some deeper connections between Barker's analysis and the computer science literature. Danvy and Filinski (1989, 1990), among others, modeled *delimited contexts* using *composable continuations*, which they manipulated using two operators *shift* and *reset*. The denotation EVERYONE (16) can be defined in terms of *shift*. To encode scope islands, Barker implicitly used *reset*.

Continuations are related to the category-theoretic concept of *monads*. Both concepts have been used to study computational side effects in programming languages (Wadler 1997 and many others), and monads have been used to characterize semantic complications in natural languages (Shan 2001). Filinski (1999) proved that, in a certain sense, composable continuations can simulate monads.

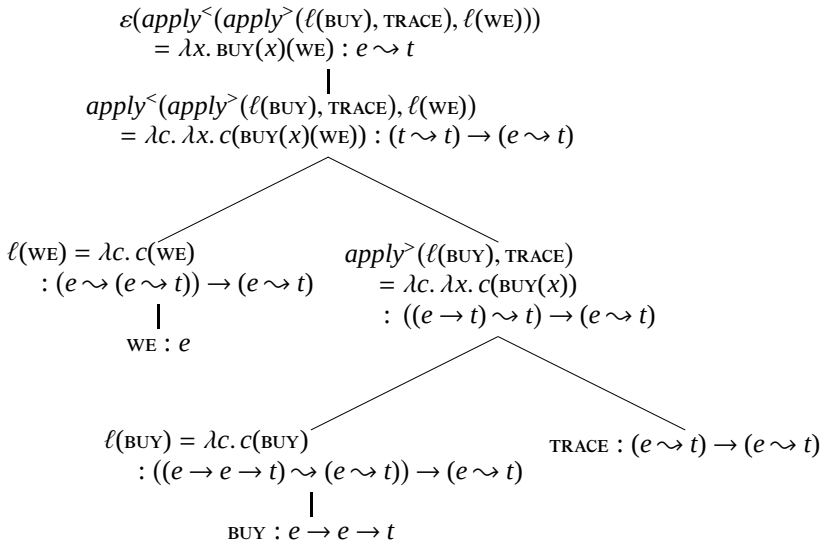

FIGURE 5. *We bought t*

3. MANIPULATING ANSWER TYPES

Having set up the framework of continuation semantics, I will now proceed to my analysis of interrogatives. I will begin by analyzing extraction and raised wh-phrases, then turn to in-situ wh-phrases and multiple-wh interrogatives.

3.1. **Extraction.**

In order to analyze interrogatives with raised wh-phrases, I first need a theory of extraction. Fortunately, continuation semantics provides for a natural and compositional analysis of extraction. One possible implementation is to posit a phonologically null NP[7] whose denotation is

$$\textsc{trace} = \lambda c.\, c : (e \rightsquigarrow \gamma) \to (e \rightsquigarrow \gamma), \tag{17}$$

that is, the identity function over individual-taking continuations.[8] I notate this element as "t". Figure 5 shows how to derive a meaning for *we bought t*. The final denotation has type $e \rightsquigarrow t$, a continuation type. This is typical of a clause with an unsaturated gap, and is intuitive considering that continuations are supposed to model evaluation contexts, in other words expressions with holes such as *we bought* ____.

Figure 5 reflects two important aspects in which the continuation semantics I presented in §2.3 nontrivially generalizes Barker's system. First, as mentioned before, my types distinguish continuations ($\rightsquigarrow$) from ordinary functions ($\to$), whereas Barker's do not. Second, Barker specified the same answer type, namely t, for all continuations and continuized values. In other words, he effectively substituted t everywhere the type variable γ occurred

[7]That I posit a phonologically null element participating in syntax is a matter of presentation and not critical to the approach to extraction sketched here. It would work equally well for my purposes to introduce type-shift operations that effectively roll TRACE into the binary composition rules in (14).

[8]The idea that a variable or gap is in some sense an identity function over continuations appeared in the work of Danvy and Filinski (1989, §3.4), and also has been mentioned to me by Barker in conversation. The more general idea that a variable or gap is an identity function of some sort has an even longer history in the literature (Hindley and Seldin 1986; Jacobson 1999, 2000).

in §2.3 (including subscripted versions). Consequently, Barker's analysis has no polymorphic denotations.

My present purposes require that we not fix a single answer type, for two reasons. First, not all clauses have the same type. Although basic declarative sentences such as *John loves Mary* have type t, I want sentences with gaps—and as we will soon see, interrogatives as well—to have other semantic types. Second, my semantics not only calls for a mixture of different answer types, but in fact contains denotations that actively modify what can be thought of as the "current answer type".

Active manipulation of answer types is exemplified by the definition of TRACE in (17) and the derivation of *we bought t* in Figure 5. Examining the definition of TRACE, we see that it takes as input an e-taking continuation whose answer type is γ, but returns a final answer of type $e \rightsquigarrow \gamma$ instead. Informally speaking, TRACE acts like an e locally, but in addition prepends "$e \rightsquigarrow$" to the current answer type; hence *we bought t* receives the semantic type $e \rightsquigarrow t$ rather than t. In general, a denotation of type

$$(\tau \rightsquigarrow \omega) \to \omega' \tag{18}$$

(where τ, ω and ω' are type expressions and may contain type variables) acts like a τ, but in addition transforms the current answer type from ω to ω'. The denotation TRACE is a special case where $\tau = e$, $\omega = \gamma$, and $\omega' = e \rightsquigarrow \gamma$. Another special case is denotations lifted using the ℓ operator (10), for which $\omega = \omega'$ and active manipulation of answer types degenerates into passive propagation.[9]

3.2. **An abbreviated notation for continuized types.** Before our types get too unwieldy, I would like to introduce an alternative, abbreviated notation for types of the form (18). From the discussion in §3.1, we know that we can think of a type $(\tau \rightsquigarrow \omega) \to \omega'$ informally as a continuized τ that changes the answer type from ω to ω'. Accordingly, I will write

$$\tau \frac{\omega}{\omega'}$$

as shorthand for $(\tau \rightsquigarrow \omega) \to \omega'$. For example, TRACE can be alternatively defined as

$$\textsc{trace} = \lambda c.\, c : e \frac{\gamma}{e \rightsquigarrow \gamma}, \tag{17$'$}$$

so as to emphasize the intuition that it acts locally like an e, but prepends "$e \rightsquigarrow$" to the answer type.

For reference, Figure 6 repeats the rules for continuizing grammars given in §2.3, writing the types in the abbreviated notation. (In the figure, notice how each of the binary rules (13a$'$) and (13b$'$) concatenates two changes to the answer type—first from γ_1 to γ_2, and then from γ_2 to γ_3—into a change from γ_1 to γ_3.) Figure 7 repeats the analysis of *we bought t* from §3.1 (Figure 5) in the abbreviated notation.

3.3. **Raised wh-phrases.** As I alluded to in §1, my interrogative denotations are—roughly speaking—functions mapping answers to propositions. To make this idea precise, I introduce yet another binary type constructor $\multimap$, so as to form *question types* such as

$$e \multimap t.$$

As before, I distinguish between the question type $e \multimap t$, the continuation type $e \rightsquigarrow t$, and the function type $e \to t$, even though they may have the same models and I overload the same λ notation for all three kinds of abstractions.

[9]This pseudo-operational description is merely an intuitive sketch. The denotations in my semantics are computed purely in-situ according to local composition rules.

(10′) $$\ell(x) = \lambda c.\, c(x) : \alpha\frac{\gamma}{\gamma}$$
$$x : \alpha$$

(11′) $$\varepsilon(q) = q(\lambda x.\, x) : \gamma$$
$$q : \alpha\frac{\alpha}{\gamma}$$

(12′) $$f^*(q) = \lambda c.\, q(\lambda x.\, c(f(x))) : \tau'\frac{\gamma_1}{\gamma_2}$$
$$q : \tau\frac{\gamma_1}{\gamma_2}$$

(13a′) $$g^{>}(q, r) = \lambda c.\, q(\lambda x.\, r(\lambda y.\, c(g(x, y)))) : \tau'\frac{\gamma_1}{\gamma_3}$$
$$q : \tau_1\frac{\gamma_2}{\gamma_3} \qquad r : \tau_2\frac{\gamma_1}{\gamma_2}$$

(13b′) $$g^{<}(q, r) = \lambda c.\, r(\lambda y.\, q(\lambda x.\, c(g(x, y)))) : \tau'\frac{\gamma_1}{\gamma_3}$$
$$q : \tau_1\frac{\gamma_1}{\gamma_2} \qquad r : \tau_2\frac{\gamma_2}{\gamma_3}$$

FIGURE 6. Continuizing Montague grammar (in abbreviated notation)

$$\varepsilon(\mathit{apply}^{<}(\mathit{apply}^{>}(\ell(\textsc{buy}), \textsc{trace}), \ell(\textsc{we}))) = \lambda x.\, \textsc{buy}(x)(\textsc{we}) : e \rightsquigarrow t$$
$$\mathit{apply}^{<}(\mathit{apply}^{>}(\ell(\textsc{buy}), \textsc{trace}), \ell(\textsc{we})) = \lambda c.\, \lambda x.\, c(\textsc{buy}(x)(\textsc{we})) : t\frac{t}{e\rightsquigarrow t}$$
$$\ell(\textsc{we}) = \lambda c.\, c(\textsc{we}) : e\frac{e\rightsquigarrow t}{e\rightsquigarrow t} \qquad \mathit{apply}^{>}(\ell(\textsc{buy}), \textsc{trace}) = \lambda c.\, \lambda x.\, c(\textsc{buy}(x)) : (e \to t)\frac{t}{e\rightsquigarrow t}$$
$$\textsc{we} : e$$
$$\ell(\textsc{buy}) = \lambda c.\, c(\textsc{buy}) : (e \to e \to t)\frac{e\rightsquigarrow t}{e\rightsquigarrow t} \qquad \textsc{trace} : e\frac{t}{e\rightsquigarrow t}$$
$$\textsc{buy} : e \to e \to t$$

FIGURE 7. *We bought t* (in abbreviated notation)

I will now treat the sentence

(25) John remembers what [we bought t].

On the way, I will also achieve an analysis of

(26) What did we buy t?

because it appears as part of (25) except for the subject-auxiliary inversion triggered by direct (top-level) interrogatives, which I ignore.

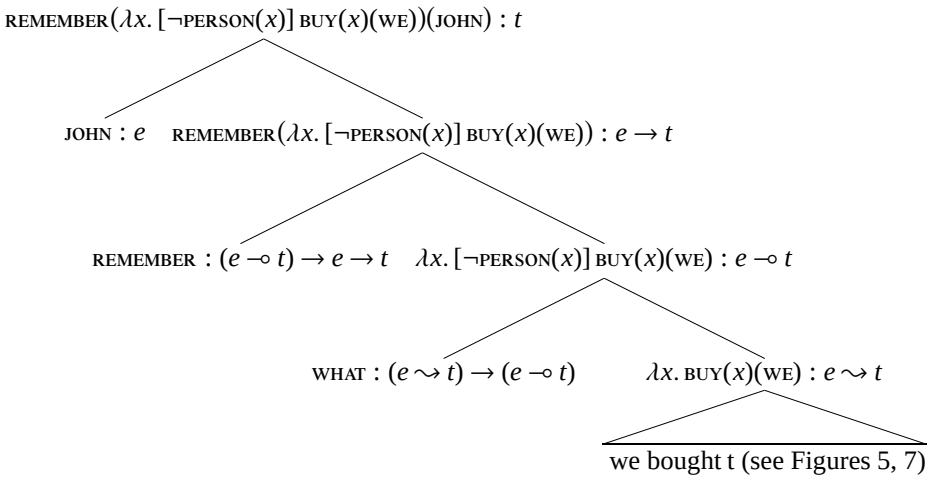

FIGURE 8. *John remembers what* [*we bought t*]

In §3.1, I had already derived a denotation of type $e \rightsquigarrow t$ for the embedded clause *we bought t*. Let us assume, as is commonly done, that remembrance is a relation between persons (type e) and questions (type $e \multimap t$, for the time being). Then, *remember* has some denotation

(27) $$\textsc{remember} : (e \multimap t) \to e \to t.$$

Having assigned meanings to every other word in (25), I am now ready to specify what *what* means. Note that *we bought t* is of type $e \rightsquigarrow t$, but *remember* requires the (distinct!) type $e \multimap t$ for its input. Therefore, the denotation of *what* should convert *we bought t* from $e \rightsquigarrow t$ to $e \multimap t$. I make the simplest assumption to that effect—that *what* has the semantic type

$$(e \rightsquigarrow t) \to (e \multimap t).$$

The semantic content of *what* should be essentially the identity function, but somehow express the requirement that the incoming answer—the input to $e \multimap t$—be nonhuman in some sense. I am not concerned with the syntactic or semantic nature of this requirement here, so I will simply notate it as a bracketed formula $[\neg\textsc{person}(x)]$, as in

(28a) $$\textsc{what} = \lambda c.\, \lambda x.\, [\neg\textsc{person}(x)]\, c(x) : (e \rightsquigarrow t) \to (e \multimap t),$$

(28b) $$\textsc{who} = \lambda c.\, \lambda x.\, [\textsc{person}(x)]\, c(x) \quad : (e \rightsquigarrow t) \to (e \multimap t).$$

One way to think of the formula $[\neg\textsc{person}(x)]\, c(x)$ intuitively is "if $\neg\textsc{person}(x)$ then $c(x)$, otherwise undefined".

The definitions in (28) makes no concrete use of the base type t. Indeed, we can generalize them to

(29a) $$\textsc{what} = \lambda c.\, \lambda x.\, [\neg\textsc{person}(x)]\, c(x) : (e \rightsquigarrow \gamma) \to (e \multimap \gamma),$$

(29b) $$\textsc{who} = \lambda c.\, \lambda x.\, [\textsc{person}(x)]\, c(x) \quad : (e \rightsquigarrow \gamma) \to (e \multimap \gamma).$$

I will make use of this generalization in a moment. For now, I have completed my analysis of (25). The derivation is shown in Figure 8. Once the meaning of *we bought t* is derived, the rest of the composition process consists entirely of simple function application.

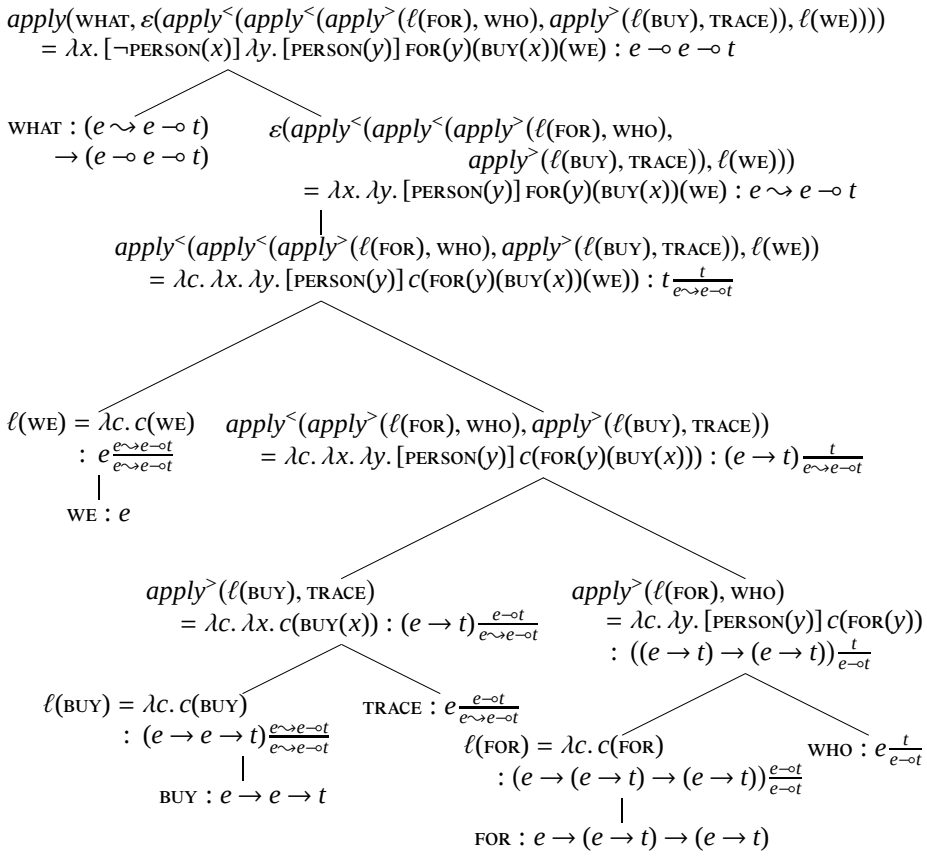

FIGURE 9. *What we bought t for whom,* with the narrow-scope reading for *whom* (in abbreviated notation)

At this point I can explain why I distinguish between function types (→), continuation types (⇝), and question types (⊸). Besides clarifying the notation, the distinction prevents the grammar from overgenerating sentences such as **I remember John bought* or **I remember what what John bought*. The semantic types naturally enforce a one-to-one correspondence between gaps and wh-phrases—more precisely, interrogatives gaps and raised wh-phrases.

3.4. **In-situ wh-phrases.** The analyses above of extraction and raised wh-phrases are both natural in the spirit of continuation semantics. The primary payoff from these analyses is that little more needs to be said to treat interrogatives with in-situ wh-phrases and to account for the two properties I listed in §1.

With respect to the first property (that wh-phrases appear both raised and in-situ), consider for instance the clauses in (30), both of which contain *what* raised and *whom* in-situ.

(30) a. What did we buy t for whom?
b. what we bought t for whom

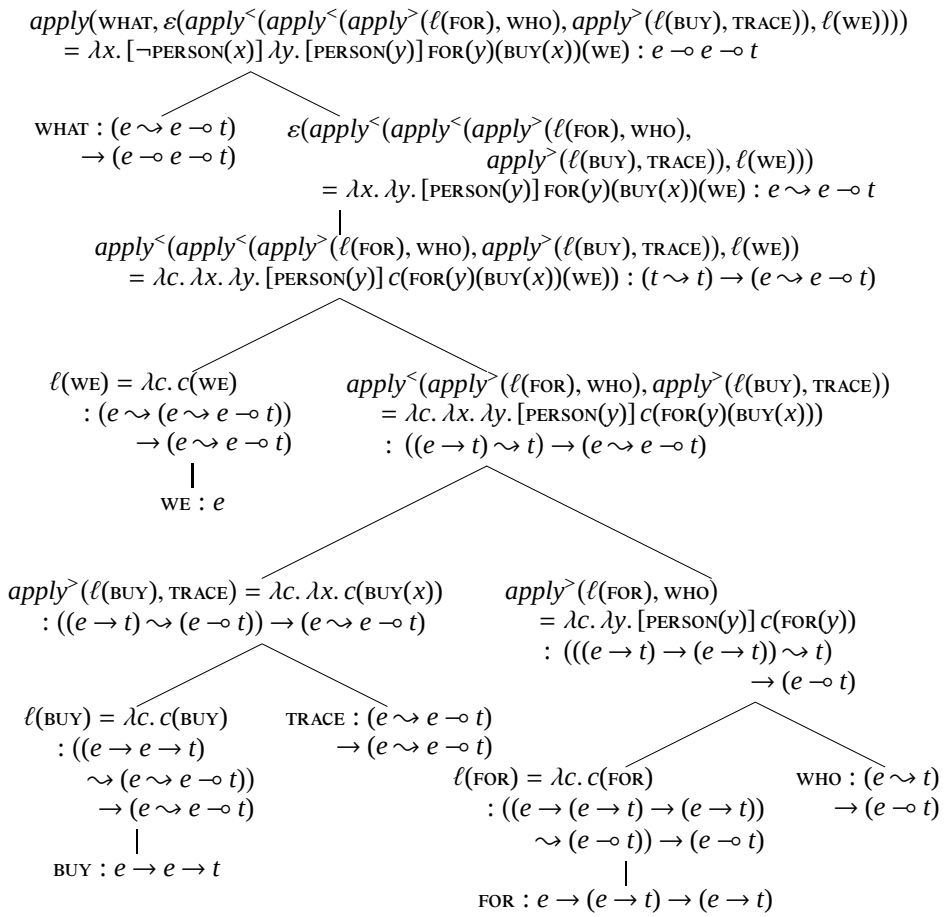

FIGURE 10. *What we bought t for whom*, with the narrow-scope reading for *whom* (in unabbreviated notation)

Ignoring the subject-auxiliary inversion in (30a), these two clauses are identical. To derive a denotation, all that is needed is an uncontroversial meaning for *for*:

(31) $$\textsc{for} : e \to (e \to t) \to (e \to t).$$

Given that it was with raised usage in mind that we assigned to *whom* its meaning in (29b), it may come as a surprise that the same meaning works equally well for in-situ usage. But it does all work out: The derivation, which culminates in the top-level type $e \multimap e \multimap t$, is shown in Figures 9 (in abbreviated notation) and 10 (in unabbreviated notation). (If we assume furthermore that *remember* can take semantic type $(e \multimap e \multimap t) \to e \to t$, then it is straightforward to produce the narrow-scope reading (3b) of the example (1b) from §1.)

To see how this derivation works, it is useful to examine (29), where *what* and *who* were assigned denotations of the (polymorphic) type

(32) $$(e \rightsquigarrow \gamma) \to (e \multimap \gamma).$$

In §3.3, this type was justified because *what* needed to convert ($\to$) an e-taking continuation ($\rightsquigarrow$) to an e-wondering question ($\multimap$). However, the discussion in §3.1 enables us to view

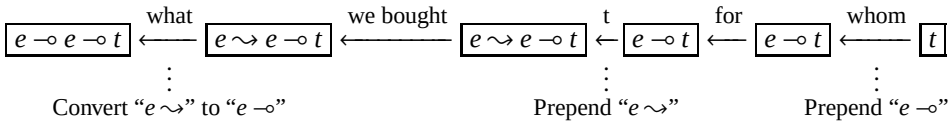

FIGURE 11. Building and assigning the semantic type $e \multimap e \multimap t$ to *what we bought t for whom*, with the narrow-scope reading for *whom*. Whereas the in-situ elements *t* and *whom* manipulate the answer type, the raised element *what* manipulates the value type.

the same type in a different way: Note that the type (32) can also be written as

$$e\frac{\gamma}{e \multimap \gamma}$$

in the notation of §3.2. A denotation of this type takes as input an *e*-taking continuation whose answer is γ, but returns a final answer of type $e \multimap \gamma$ instead. Informally speaking, then, interrogative NPs such as *what* and *who* act like *e*s locally, but in addition prepends "$e \multimap$" to the current answer type.

Hence, as one might expect, the double-wh constructions in (30) receive the semantic type $e \multimap e \multimap t$. The first "$e \multimap$" in the type is contributed by the raised wh-phrase *what*, or rather, contributed as "$e \rightsquigarrow$" by the extraction gap and subsequently converted to "$e \multimap$" by *what*. The second "$e \multimap$" in the type is contributed directly by the in-situ wh-phrase *whom*. Figure 11 illustrates this process.

It might be initially perplexing why the types in Figure 11 are manipulated *right-to-left*. This pattern is naturally explained if we postulate that evaluation in natural language tends to proceed *left-to-right* in terms of the linear order of constituents. The notion of evaluation order I refer to here came up in §2.3, where I described how, when a grammar is continuized, each binary composition method gives rise to two new ones that differ in evaluation order. In particular, $apply^{>}$ (14a) evaluates the function before the argument, while $apply^{<}$ (14b) evaluates the argument before the function. The idea of left-to-right evaluation can thus be rephrased as follows: When a function occurs before its argument, $apply^{>}$ is preferred; when the function occurs after its argument, $apply^{<}$ is preferred. For consistency, every derivation I show in this paper uses left-to-right evaluation throughout, but other evaluation orders are also possible and give rise to equivalent overall meanings.

Left-to-right evaluation results in right-to-left answer type manipulation, because what left-to-right evaluation means is that the constituents to the left get to decide first what the answer type looks like at outer levels. For example, TRACE wants the answer type to be a continuation at the outermost level it gets to affect. When building the answer type bottom-up from *t* to $e \multimap e \multimap t$, the outermost decisions are executed last, not first.

Critical to the ability of the type $(e \rightsquigarrow \gamma) \to (e \multimap \gamma)$ to serve two roles at once is the type-shift operator ε. In-situ wh-phrases (and gaps) combine with other constituents and manipulate the answer type through either $apply^{>}$ or $apply^{<}$. By contrast, raised wh-phrases combine with other constituents through good old function application, and perform the conversion from "$e \rightsquigarrow$" to "$e \multimap$" not on promised answers but on evaluated values. Before a raised wh-phrase can act on a gapped clause, then, the clause needs a meaning whose value type—*not answer type*—is of the form $e \rightsquigarrow \cdots$. The ε operator fills this need: It extracts an answer out of a continuized value by feeding it the identity continuation.

3.5. **Wide scope and Baker's ambiguity.** We have seen above that, in my analysis of interrogatives, a single denotation for each wh-phrase suffices for both raised and in-situ appearances, as long as the wh-phrase takes narrow scope. To fulfill the promises I made in §1, I have to turn to wide scope and account for two additional facts about interrogatives: First, I have to show in my system that in-situ wh-phrases can take semantic scope wider than the immediately enclosing clause, as they do in my initial examples (2a) and (3a). Second, I have to show that raised wh-phrases cannot take wide scope; in other words, they must take semantic scope exactly where they are overtly located.

I claim to account for wide scope interrogatives using *higher-order continuations*, in other words by continuizing the grammar *twice*.[10] Starting from "pure" Montague grammar (i.e., only function application), recall that continuizing once adds two type-shift operators (ℓ and ε) and two binary composition rules (*apply*$^{>}$ and *apply*$^{<}$). As illustrated in the rightmost column of Figure 2, continuizing again adds two more type-shift operators (ℓ^* and ε^*) and four more binary composition rules (*apply*$^{>>}$, *apply*$^{><}$, *apply*$^{<>}$, and *apply*$^{<<}$).

Before plunging into semantic derivations, we first need some intuitive understanding of how twice-continuized grammars work. In a once-continuized grammar, many types are of the form

$$(\tau \rightsquigarrow \omega) \to \omega', \quad \text{or equivalently,} \quad \tau \frac{\omega}{\omega'}, \tag{33}$$

where τ, ω and ω' are some types. In §3.1, I explained how we can view such a type informally to mean "acts locally like a τ, but in addition transforms the answer type from ω to ω'". In a twice-continuized grammar, many types are of the form

$$(((\tau \rightsquigarrow \omega) \to \omega') \rightsquigarrow \varpi) \to \varpi', \quad \text{or equivalently,} \quad \tau \frac{\omega}{\omega'} \frac{\varpi}{\varpi'}, \tag{34}$$

where τ, ω, ω', ϖ, and ϖ' are some types. One way to understand such types is to think of a derivation in a twice-continuized grammar as maintaining two answer types—an *inner* answer type corresponding to the first continuization, and an *outer* one corresponding to the second. A type of the form (34) means "acts locally like a τ, but in addition transforms the inner answer type from ω to ω', and the outer answer type from ϖ to ϖ'".

To strengthen this understanding, let us examine how answer types are manipulated in the four binary composition rules that result from continuizing *apply* twice. In (35) below, I have expanded out the rules, in particular the types.

(35a)
$$\mathit{apply}^{>>}(u,v) = \lambda d.\, u(\lambda q.\, v(\lambda r.\, d(\lambda c.\, q(\lambda x.\, r(\lambda y.\, c(x(y))))))) : \beta \frac{\gamma_1}{\gamma_3} \frac{\delta_1}{\delta_3}$$
$$u : (\alpha \to \beta) \frac{\gamma_2}{\gamma_3} \frac{\delta_2}{\delta_3} \qquad v : \alpha \frac{\gamma_1}{\gamma_2} \frac{\delta_1}{\delta_2}$$

(35b)
$$\mathit{apply}^{><}(u,v) = \lambda d.\, v(\lambda r.\, u(\lambda q.\, d(\lambda c.\, q(\lambda x.\, r(\lambda y.\, c(x(y))))))) : \beta \frac{\gamma_1}{\gamma_3} \frac{\delta_1}{\delta_3}$$
$$u : (\alpha \to \beta) \frac{\gamma_2}{\gamma_3} \frac{\delta_1}{\delta_2} \qquad v : \alpha \frac{\gamma_1}{\gamma_2} \frac{\delta_2}{\delta_3}$$

[10] In the same spirit, Barker (2000b) used higher-order continuations to treat wide-scope specific indefinites and some interactions between coordination and antecedent-contained deletion.

(35c)
$$\mathit{apply}^{>>}(u,v) = \lambda d.\, u(\lambda q.\, v(\lambda r.\, d(\lambda c.\, r(\lambda y.\, q(\lambda x.\, c(x(y))))))) : \beta\frac{\gamma_1}{\gamma_3}\frac{\delta_1}{\delta_3}$$
$$u : (\alpha \to \beta)\frac{\gamma_1}{\gamma_2}\frac{\delta_2}{\delta_3} \qquad v : \alpha\frac{\gamma_2}{\gamma_3}\frac{\delta_1}{\delta_2}$$

(35d)
$$\mathit{apply}^{><}(u,v) = \lambda d.\, v(\lambda r.\, u(\lambda q.\, d(\lambda c.\, r(\lambda y.\, q(\lambda x.\, c(x(y))))))) : \beta\frac{\gamma_1}{\gamma_3}\frac{\delta_1}{\delta_3}$$
$$u : (\alpha \to \beta)\frac{\gamma_1}{\gamma_2}\frac{\delta_1}{\delta_2} \qquad v : \alpha\frac{\gamma_2}{\gamma_3}\frac{\delta_2}{\delta_3}$$

In the rules $\mathit{apply}^{>>}$ and $\mathit{apply}^{><}$, evaluation proceeds left-to-right (that is, from function to argument) at the first continuation level. Accordingly, the subscripts on the γs show that the inner answer type is threaded first through the right hand side v and then through the left hand side u. In the rules $\mathit{apply}^{<>}$ and $\mathit{apply}^{<<}$ the reverse happens: Evaluation proceeds right-to-left (that is, from argument to function), and the inner answer type is threaded through first u and then v.

Similarly for the second continuation level: In the rules $\mathit{apply}^{>>}$ and $\mathit{apply}^{<>}$, evaluation proceeds from function to argument, and the subscripts on the δs show that the outer answer type is threaded first through v and then through u. In the rules $\mathit{apply}^{><}$ and $\mathit{apply}^{<<}$, evaluation proceeds from argument to function, and the outer answer type is threaded through first u and then v.

With these remarks in mind, let us turn to some semantic derivations. Our goal is to compute the wide-scope reading (3a) for the sentence (1b) from §1, repeated here with gaps represented explicitly.

(1b′) Who do you think t remembers [what we bought t for whom]?

I will add no new denotations to the lexicon other than the obvious missing entries

$$\textsc{you} : e, \qquad \textsc{think} : t \to e \to t,$$

and extend the grammar only by continuizing it for a second time as discussed above.

The wide-scope reading of the example (1b) is a double-wh question. Thus, we expect the matrix clause to have the semantic type $e \multimap e \multimap t$. What about the embedded clause? It acts locally like a single-wh question (*what*), but in addition should prepend "$e \multimap$" to the answer type (*whom*). Therefore, we expect the embedded clause to have the type

(36)
$$(e \multimap t)\frac{\delta}{e \multimap \delta},$$

which includes the special case

(37)
$$(e \multimap t)\frac{t}{e \multimap t}.$$

Figure 12 shows a derivation for *what we bought t for whom* that culminates in precisely the type (36). The interesting part of the derivation is how the three continuation-manipulating elements *what*, *t*, and *whom* enter it. The narrow-scope elements *what* and *t* need to manipulate the inner answer type while remaining oblivious to the second continuation level; to achieve this effect, we lift them to $\ell(\textsc{what})$ and $\ell(\textsc{trace})$, respectively, and they participate in the derivation as such. The wide-scope element *whom*, on the other hand, needs to manipulate the outer answer type while remaining oblivious to the first continuation level; to achieve this effect, we lift it "from the inside" using ℓ^*, a type-shift operator that became available when the grammar was continuized for the second time.

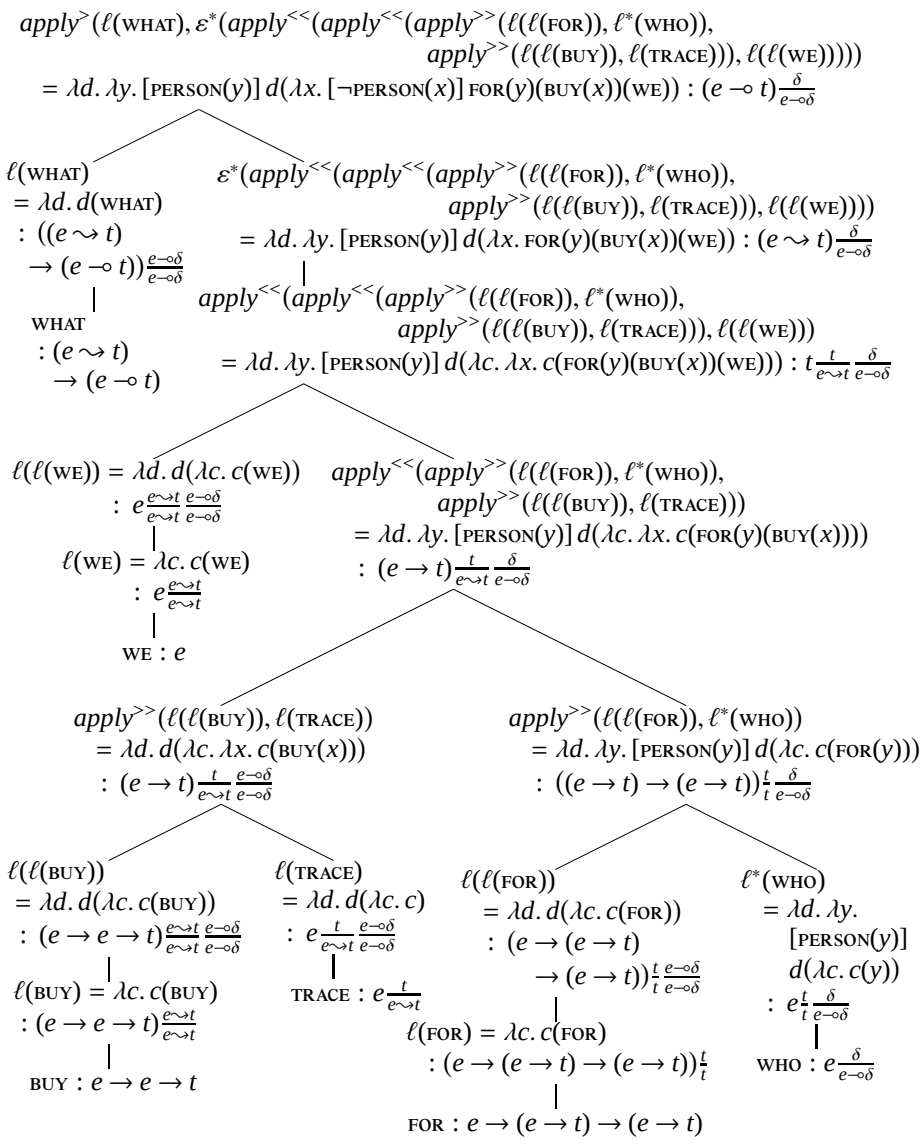

FIGURE 12. *What we bought t for whom*, with the wide-scope reading for *whom*

Near the top of the derivation tree, we invoke the operator ε^* rather than ε to evaluate the embedded clause "from the inside", i.e., on the first continuation level rather than the second.

Given a meaning for *what we bought t for whom* that is of the required type (37), the semantics of the matrix question follows easily from the techniques already demonstrated in previous sections. Second-order continuations are no longer involved. The embedded clause is just a constituent that contains an in-situ wh-phrase *whom*; like any other such constituent, it combines with the rest of the sentence, including the raised wh-phrase *who*, to give a final denotation of type $e \multimap e \multimap t$, typical of a double-wh interrogative. I omit the

derivation tree and give only the top-level result:

$$\begin{aligned}
&\mathit{apply}(\textsc{what}, \varepsilon(\mathit{apply}^{<}(\mathit{apply}^{>}(\ell(\textsc{think}),\\
&\qquad \mathit{apply}^{<}(\mathit{apply}^{>}(\ell(\textsc{remember}), \sim), \textsc{trace})), \ell(\textsc{you}))))\\
&= \lambda z.\, [\textsc{person}(z)]\, \lambda y.\, [\textsc{person}(y)]\\
&\qquad \textsc{think}(\textsc{remember}(\lambda x.\, [\neg\textsc{person}(x)]\, \textsc{for}(y)(\textsc{buy}(x))(\textsc{we}))(z))(\textsc{you})\\
&: e \multimap e \multimap t.
\end{aligned} \tag{38}$$

Here the symbol $\sim$ stands for the denotation of the embedded clause derived in Figure 12.

I have demonstrated that higher-order continuations allow in-situ wh-phrases to take wide scope. It now remains for me to explain why raised wh-phrases cannot take wide scope,[11] no matter how many times we continuize the grammar. What does it mean for a raised wh-phrase to take narrow scope versus wide scope? Based on the analyses so far, I make the following definitional characterization: A raised wh-phrase takes narrow scope when it contributes its "$e \multimap$" to the clause's value type, and wide scope when it contributes its "$e \multimap$" to the clause's outgoing answer type (or rather, one of the clause's outgoing answer types, in the presence of higher-order continuations).

To be more precise, suppose that we have some clause with a raised wh-phrase in front and a corresponding gap inside.

(39) $[_a\, [_b\, \text{wh}]\, [_c \ldots\, [_d\, \text{t}]\, \ldots]]$

The defining characteristic for the wh-phrase b to take narrow scope is for the clause a to have semantic type of the form

$$(e \multimap \langle\text{some type}\rangle)\, \frac{\overbrace{\langle\text{some type}\rangle}^{\text{zero or more times}}}{\langle\text{some type}\rangle}, \tag{40a}$$

and the clause-sans-wh-phrase c the corresponding form

$$(e \rightsquigarrow \langle\text{some type}\rangle)\, \frac{\overbrace{\langle\text{some type}\rangle}^{\text{zero or more times}}}{\langle\text{some type}\rangle}, \tag{40b}$$

such that the "$e \multimap$" was contributed as "$e \rightsquigarrow$" by the gap d and subsequently converted to "$e \multimap$" by the wh-phrase b. On the other hand, for the wh-phrase b to take wide scope means for a to have semantic type of the form

$$\langle\text{some type}\rangle \frac{\langle\text{some type}\rangle}{e \multimap \langle\text{some type}\rangle}\, \frac{\overbrace{\langle\text{some type}\rangle}^{\text{zero or more times}}}{\langle\text{some type}\rangle}, \tag{41a}$$

and c the corresponding form

$$\langle\text{some type}\rangle \frac{\langle\text{some type}\rangle}{e \rightsquigarrow \langle\text{some type}\rangle}\, \frac{\overbrace{\langle\text{some type}\rangle}^{\text{zero or more times}}}{\langle\text{some type}\rangle}, \tag{41b}$$

such that the "$e \multimap$" was contributed as "$e \rightsquigarrow$" by d and subsequently converted to "$e \multimap$" by b. I wish to show the latter case impossible.

[11] Note that I do not mean that a raised wh-phrase cannot take scope beyond the clause immediately enclosing its corresponding gap. What I mean is that a raised wh-phrase cannot take scope beyond the clause in front of which it is pronounced.

The latter case is impossible because transforming the type (41b) to the type (41a) requires manipulating an answer type to replace "$e \rightsquigarrow$" with "$e \multimap$", a feat neither any element in the lexicon nor any rule in the grammar accomplishes: A quick survey of the lexical items and grammar rules in this paper reveals that the they and their descendants by continuization only manipulate answer types by adding to them. Nothing ever removes or replaces components of any answer type until the answer has been lowered to value level by evaluation. The formalism I use here does not stipulate this—in fact, one can easily introduce raised wh-phrases into the lexicon that take wide scope:

(42a) $\textsc{what}' = \lambda p.\, \lambda c.\, \lambda x.\, [\neg \textsc{person}(x)]\; p(c)(x) : \alpha\dfrac{\gamma}{e \rightsquigarrow \gamma} \rightarrow \alpha\dfrac{\gamma}{e \multimap \gamma}$,

(42b) $\textsc{who}' = \lambda p.\, \lambda c.\, \lambda x.\, [\textsc{person}(x)]\; p(c)(x) \quad : \alpha\dfrac{\gamma}{e \rightsquigarrow \gamma} \rightarrow \alpha\dfrac{\gamma}{e \multimap \gamma}$.

Nevertheless, these denotations do not appear to exist in English, and in any case would not be related by any type-shift operations to WHAT and WHO as defined earlier in (29).

4. Discussion

In this paper, I presented a grammar fragment that captures two properties of interrogatives in English: First, wh-phrases can appear both raised and in-situ. Second, in-situ wh-phrases can take scope beyond the immediately enclosing clause, but raised wh-phrases must take scope exactly where they are pronounced. My fragment draws inspiration from Barker's continuation semantics for natural language (2000a, 2000b) as well as work on continuations and typed contexts in programming languages (Danvy and Filinski 1989, 1990; Murthy 1992; Wadler 1994).

As a general semantics of interrogatives, my treatment raises many concerns that I have not addressed in this paper. These include:

- languages other than English;
- pair-list questions;
- relative clauses;
- interaction with intensionality and quantification; and
- Hirschbühler's (1978) arguments against treating questions as functions from answers to propositions.

Regardless, the two properties of interrogatives that I did explore in this paper fall out of my basic analysis in a surprisingly natural way: Wh-phrases can appear both raised and in-situ because a single denotation works at both positions, but in raised position this denotation is forced to take overt scope on the basis of general, non-stipulative type-theoretic considerations. The basic ideas probably carry over to other kinds of so-called A′-movement, such as topicalization. I explore some directions below.

4.1. **Superiority.**

Near the end of §3.4, I suggested that evaluation in natural language tends to proceed left-to-right in terms of the linear order of constituents. More generally, there may be constraints or processing preferences that govern evaluation order in natural language; possible factors include linear precedence and discourse linking. Such patterns in evaluation order may explain superiority effects. For example, if we eliminate all but left-to-right evaluation and first-order continuations from our grammar, then the sentence (43a) below remains derivable, but its superiority-violating counterpart (43b) is no longer admitted.

(43) a. Who bought what?
b. *What did who buy? (non-echo-question reading)

The reason is once again type-theoretic: The constituent *who bought t* can take semantic type $e \multimap e \rightsquigarrow t$ but not $e \rightsquigarrow e \multimap t$, and so cannot combine with WHAT : $(e \rightsquigarrow \gamma) \to (e \multimap \gamma)$.

4.2. **Pied-piping.** My system does not explain when pied-piping is required, allowed, or prohibited. As is, though, it already admits sentences with pied-piping. For example, we can derive the same meaning for all four sentences in (44). This is desirable for the grammatical (44a) and (44b), but undesirable for the ungrammatical (44c) and (44d).

(44) a. With whose authority did you command these troops t?
b. Whose authority did you command these troops with t?
c. *Whose did you command these troops with t authority?
d. *Command these troops with whose authority did you t?

Take for example the sentence (44b), whose syntactic structure we assume to be (45).

(45) [Whose authority] [did you command these troops with t]

The wh-phrase *whose* acts locally as value type $(e \to t) \to e$, but in addition prepends "$e \multimap$" to the current answer type. In other words, its denotation WHOSE has the semantic type

(46) $$((e \to t) \to e)\frac{\gamma}{e \multimap \gamma}.$$

Treating *whose* as an in-situ wh-phrase with respect to *authority*, we can derive for *whose authority* the meaning

$$\mathit{apply}^{>}(\textsc{whose}, \textsc{authority}),$$

which is of type

$$e\frac{\gamma}{e \multimap \gamma}, \quad \text{or equivalently,} \quad (e \rightsquigarrow \gamma) \to (e \multimap \gamma).$$

This type for *whose authority* is identical to our type for *what* (29). This is appropriate given the structural similarity between (45) and (47).

(47) a. What did you command these troops with?
b. [What] [did you command these troops with t]

From *whose authority* onward, then, we can treat (45) exactly as we treat (47). No additional stipulation in the grammar is necessary.

The other three sentences in (44) can be analyzed analogously. Note that the extracted constituents have type $(e \to t) \to (e \to t)$ in (44a), $(e \to t) \to e$ in (44c), and $e \to t$ in (44d), rather than e as in (44b). Accordingly, we need to generalize the type of TRACE from

$$(e \rightsquigarrow \gamma) \to (e \rightsquigarrow \gamma),$$

as given in (17), to

$$(\alpha \rightsquigarrow \gamma) \to (\alpha \rightsquigarrow \gamma).$$

The grammar can then generate all four sentences in (44).[12]

[12]Rather than generalizing TRACE all the way to allow any semantic type to be extracted, we could add just one additional version of TRACE that only allows extracting the type $(e \to t) \to (e \to t)$. Such an approach would account for the judgments in (44), at the expense of stipulating multiple versions of TRACE. I do not expect restrictions on what semantic types can be extracted to completely explain where pied-piping occurs.

4.3. **Topicalization.** My analysis of extraction already admits sentences with topicalization.

(48) a. The vase, John bought.
b. [The vase] [John bought t]

The derivation is straightforward: Use ℓ to lift the semantic type of *the vase* from e to $(e \rightsquigarrow t) \to t$, then *apply* the result to *John likes t very much*, which has semantic type $e \rightsquigarrow t$.

Topicalization interacts with interrogatives. In English, wh-phrases cannot topicalize (49) (Lasnik and Uriagereka 1988, page 15), but in Chinese they can (50) (Chen 2001a, 2001b).

(49) a. Who remembers we bought what? (John remembers we bought the vase.)
b. *Who remembers what we bought? (John remembers we bought the vase.)

(50) a. shei jide women mai-le sheme?
who remember we buy-ASP what
'Who remembers we bought what?' (John remembers we bought the vase.)
b. shei jide sheme women mai-le?
who remember what we buy-ASP
'Who remembers we bought what?' (John remembers we bought the vase.)

Unfortunately for English, my semantics does generate topicalized wh-phrases. For example, it fails to rule out (49b). To see this, note that ℓ^*(WHAT) has type

$$e\frac{t}{t}\frac{\gamma}{e \multimap \gamma}, \quad \text{or equivalently,} \quad ((e \rightsquigarrow t) \to t)\frac{\gamma}{e \multimap \gamma}.$$

Thus, we can *apply*$^>$ it to *we bought t*, which as before has semantic type $e \rightsquigarrow t$. The final result has type

$$t\frac{\gamma}{e \multimap \gamma},$$

as expected.[13]

4.4. **Interrogative denotations.** I have described a semantics of interrogatives in which interrogative clauses denote functions from answers to propositions. One reason to expect this semantics to bear on other kinds of so-called apparent movement is that it does not actually rely on interrogative clauses denoting such functions.

A prominent example is the many semantic treatments of interrogatives, starting with that of Karttunen (1977), that let interrogative clauses denote sets of propositions. It is easy to adapt my system to such a view while preserving its account of raised versus in-situ wh-phrases. Simply

- replace each type of the form $\tau_1 \multimap \tau_2 \multimap \cdots \multimap \tau$ with $\tau \to t$, the type of τ-sets;
- add to the lexicon a silent morpheme $\text{Q} = \lambda p.\{p\} : t \to t \to t$; and
- adjust the denotations WHO and WHAT accordingly.

Therefore, as I stated in §1, the continuation semantics in this paper sheds light not so much on what interrogatives denote but on the combinatorics that govern their composition.

[13] It may seem that my grammar has generated a raised wh-phrase taking scope beyond where it is pronounced, contradicting my argument at the end of §3.5. In fact, what my grammar has generated is an in-situ wh-phrase that happens to be the fronted constituent in a topicalization construction. The types show this: Note that ℓ^*(WHAT) does not convert any "$e \rightsquigarrow$" to "$e \multimap$" in the value type, so it is not a raised wh-phrase taking narrow scope as defined in (40). Neither does it perform any such conversion in the answer type, so it is not a raised wh-phrase taking wide scope as defined in (41).

4.5. **Discontinuous constituency.** This work has been guided throughout by types, which circumscribe semantic denotations as well as regulate syntactic combination. In this paper, I have primarily treated types as constraints that are semantically motivated, but the syntactic perspective is also worth exploring.

The closest syntactic counterpart to continuation semantics is the study of *discontinuous constituency* (Bunt and van Horck 1996). In the tradition of categorial (type-logical) grammar, Moortgat (1996) introduced ternary type constructors of the form

$$op(A, B, C) \tag{51}$$

and suggested that they be applied in a uniform approach to "binding" phenomena such as quantification, extraction, focus, and topicalization. In Moortgat's terms, the components A, B, and C of the type (51) are the "bound expression", the "binding domain", and the "resultant expression", respectively. These components correspond to what I term the "value type", the "incoming answer type", and the "outgoing answer type", respectively. Moortgat's syntactic type (51) thus appears to correspond to my semantic type

$$A\frac{B}{C}, \quad \text{or equivalently,} \quad (A \rightsquigarrow B) \to C. \tag{52}$$

The intrinsic connection between continuation semantics and discontinuity syntax remains to be explored. For now, it is interesting to note that my semantic type (52) uses not a ternary type constructor but two binary type constructors, one of which is the ordinary function type constructor. This decomposition is crucial for the same wh-denotation such as WHAT or WHO to participate in composition both raised and in-situ.

4.6. **Typed contexts.** Curiously, the hierarchy of higher-order continuation types used here does not seem to correspond exactly to any type system in the programming languages literature. Danvy and Filinski described one system (1989) that only allows one level of continuations, then another (1990) that handles multiple levels but embeds all value types into a single universal type. Murthy's hierarchy (1992) deals with a kind of higher-order continuations different from those used here: Expressed in my notation, a twice-continuized type in Murthy's system is of the form

$$\omega\frac{\varpi}{\varpi}\,\tau\frac{\varpi}{\omega\frac{\varpi}{\varpi}} \tag{53}$$

rather than the form

$$\tau\frac{\omega}{\omega}\frac{\varpi}{\varpi}. \tag{54}$$

His system also did not allow for dynamic manipulation of answer types. For future research, Murthy did suggest (§10) constructing a call-by-name hierarchy, as opposed to the call-by-value one he examined. Such a hierarchy would be closer to the system in this paper. In sum, it remains to be explored how these distinct ways of typing evaluation contexts relate to apparent movement phenomena in natural language.

REFERENCES

Baker, C. L. (1968). *Indirect Questions in English*. Ph. D. thesis, University of Illinois.

Barker, C. (2000a, 4 November). Continuations and the nature of quantification. Manuscript, University of California, San Diego; `http://www.semanticsarchive.net/Archive/902ad5f7/`.

Barker, C. (2000b, 15 November). Notes on higher-order continuations. Manuscript, University of California, San Diego.

Bunt, H. C. and A. van Horck (Eds.) (1996). *Discontinuous Constituency*. Berlin: de Gruyter.

Chen, L. (2001a, 12 October). (Non)topicalizability of wh-phrases. LINGUIST List 12.2554, `http://www.linguistlist.org/issues/12/12-2554.html`.

Chen, L. (2001b, 17 October). Topicalization of wh-phrases. LINGUIST List 12.2601, `http://www.linguistlist.org/issues/12/12-2601.html`.

Danvy, O. (2001, 4 May). Many happy returns. Slides for talk at the 5th International Conference on Typed Lambda Calculi and Applications, `http://www.brics.dk/~danvy/Talks/many-happy-returns.pdf`.

Danvy, O. and A. Filinski (1989). A functional abstraction of typed contexts. Technical Report 89/12, DIKU, University of Copenhagen, Denmark. `http://www.daimi.au.dk/~danvy/Papers/fatc.ps.gz`.

Danvy, O. and A. Filinski (1990). Abstracting control. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, Nice, France, pp. 151–160. New York: ACM Press.

Filinski, A. (1999). Representing layered monads. In *POPL '99: Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, San Antonio, TX, pp. 175–188. New York: ACM Press.

Hausser, R. (1978). *Linguistic Cross-Connections: A Formal Fragment of English*, Chapter 5 (The logic of questions and answers), pp. 195–247. München: Institut für Deutsche Philologie, Ludwig-Maximilians-Universität.

Hausser, R. and D. Zaefferer (1979). Questions and answers in a context-dependent Montague grammar. In F. Guenthner and S. J. Schmidt (Eds.), *Formal Semantics and Pragmatics for Natural Languages*. Dordrecht: Reidel.

Hindley, J. R. and J. P. Seldin (1986). *Introduction to Combinators and λ-Calculus*. Cambridge: Cambridge University Press.

Hirschbühler, P. (1978). *The Syntax and Semantics of Wh-Constructions*. Ph. D. thesis, Department of Linguistics, University of Massachusetts.

Jacobson, P. (1999). Towards a variable-free semantics. *Linguistics and Philosophy 22*(2), 117–184.

Jacobson, P. (2000). Paycheck pronouns, Bach-Peters sentences, and variable-free semantics. *Natural Language Semantics 8*(2), 77–155.

Karttunen, L. (1977). Syntax and semantics of questions. *Linguistics and Philosophy 1*(1), 3–44.

Kelsey, R., W. Clinger, J. Rees, et al. (1998). Revised^5 report on the algorithmic language Scheme. *Higher-Order and Symbolic Computation 11*(1), 7–105. Also in *ACM SIGPLAN Notices 33*(9), 26–76.

Lasnik, H. and J. Uriagereka (1988). *A Course in GB Syntax: Lectures on Binding and Empty Categories*. Cambridge, MA: MIT Press.

Montague, R. (1974). The proper treatment of quantification in ordinary English. In R. Thomason (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, pp. 247–270. New Haven: Yale University Press.

Moortgat, M. (1996). Generalized quantification and discontinuous type constructors. See Bunt and van Horck (1996), pp. 181–207.

Murthy, C. R. (1992). Control operators, hierarchies, and pseudo-classical type systems. In O. Danvy and C. Talcott (Eds.), *CW'92: Proceedings of the ACM SIGPLAN Workshop on Continuations*, San Francisco, CA, pp. 49–71. Technical Report STAN-CS-92-1426, Department of Computer Science, Stanford University, Stanford. ftp://cstr.stanford.edu/pub/cstr/reports/cs/tr/92/1426.

Shan, C.-c. (2001). Monads for natural language semantics. In K. Striegnitz (Ed.), *Proceedings of the ESSLLI-2001 Student Session*, Helsinki, Finland, pp. 285–298. 13th European Summer School in Logic, Language and Information. http://www.digitas.harvard.edu/~ken/esslli2001/.

Steedman, M. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory 5*, 403–440.

Steedman, M. (1996). *Surface Structure and Interpretation*. Cambridge, MA: MIT Press.

Wadler, P. (1994). Monads and composable continuations. *Lisp and Symbolic Computation 7*(1), 39–56.

Wadler, P. (1997). How to declare an imperative. *ACM Computing Surveys 29*(3), 240–263.

Harvard University, 33 Oxford Street, Cambridge, MA 02138, USA
E-mail address: ken@digitas.harvard.edu