Continuation Hierarchy and Quantifier Scope

Oleg Kiselyov and Chung-chieh Shan

[1] oleg@okmij.org
[2] ccshan@indiana.edu

Abstract. We present a directly compositional and type-directed analysis of quantifier ambiguity, scope islands, wide-scope indefinites and inverse linking. It is based on Danvy and Filinski's continuation hierarchy, with deterministic semantic composition rules that are *uniquely* determined by the formation rules of the overt syntax. We thus obtain a compositional, uniform and parsimonious treatment of quantifiers in subject, object, embedded-NP and embedded-clause positions without resorting to Logical Forms, Cooper storage, type-shifting and other ad hoc mechanisms.

To safely combine the continuation hierarchy with quantification, we give a precise logical meaning to often used informal devices such as picking a variable and binding it off. Type inference determines variable names, banishing "unbound traces".

Quantifier ambiguity arises in our analysis solely because quantifier words are polysemous, or come in several strengths. The continuation hierarchy lets us assign strengths to quantifiers, which determines their scope. Indefinites and universals differ in their scoping behavior because their lexical entries are assigned different strengths. PPs and embedded clauses, like the main clause, delimit the scope of embedded quantifiers. Unlike the main clause, their limit extends only up to a certain hierarchy level, letting higher-level quantifiers escape and take wider scope. This interplay of strength and islands accounts for the complex quantifier scope phenomena.

We present an economical "direct style", or continuation hierarchy on-demand, in which quantifier-free lexical entries and phrases keep their simple, unlifted types.

1 Introduction

The proper treatment of quantification has become a large research area ever since Montague called attention to "the puzzling cases of quantification and reference" back in 1974 [1]. The impressive breadth of the area is evident from two recent surveys [2, 3], which concentrate only on interactions of quantifier phrases among themselves (leaving out, for example, binding of pronouns by quantifiers). The two surveys collect a great amount of empirical data – more and more puzzles. There is also a great number of proposals for a theory to explain the puzzles. And yet even the basic features of the theory remain undecided. In the conclusion of her survey [2] Szabolcsi poses the following three challenges that call for significant new research:

1. "develop the tools, logical as well as syntactic, that are necessary to account for the whole range of existing readings;"
2. "draw the proper empirical distinction between readings that are actually available and those that are not;"
3. determine "whether 'spell-out syntax' is sufficient for the above two purposes" [in other words, if quantifier scope can be determined without resorting to Logical Form]

This paper takes on the challenges and develops a logical tool that is expressive to capture empirical data – available and unavailable readings – for a range of quantifier phenomena, from quantifier ambiguity to scope islands, wide-scope indefinites and inverse linking. The "spell-out syntax" proved sufficient: we directly compose meanings that are model-theoretic, not trees. There is quite more work yet to do. Future work dealing with numeric and downward-entailing quantifiers, plural indefinites, and quantificational binding will hopefully clarify presently ad hoc parameters such as the number of hierarchy levels.

1.1 What is quantifier scope

"The scope of an operator is the domain within which it has the ability to affect the interpretation of other expressions" [2, §1.1]. In this paper, we concentrate on how a quantifier affects the interpretation of another quantified phrase. For example,

(1) I showed every boy a planet.

has the reading that I showed each boy a possibly different planet. The quantifier 'every' affected the interpretation of 'a planet', which refers to a possibly different planet for a different boy. That reading is called *linear* scope. The sentence has another – *inverse* – reading, whereupon each boy was shown the same planet. The example thus exhibits quantifier ambiguity. Although the inverse-scope reading of (1) entails the linear reading (which lead to doubts if inverse readings have to be accounted for at all [4]), this is not always the case. For example, the linear and inverse readings of

(2) Two of the students attended three of the seminars.

(3) Neither student attended a seminar on rectangular circles.

do not entail each other. Szabolcsi [2] demonstrates solid inverse-scope readings on many more examples. A theory of scope must also explain why no quantifier ambiguity arises in examples like

(4) That every boy left upset a teacher.

(5) Someone reported that John saw everyone.

(6) Some professor admires every student and hates the Dean.

and yet other examples with a quantifier within an embedded clause, such as

(7) Everyone reported that [Max and some lady] disappeared.

are ambiguous. Szabolcsi argues [2, §3.2] that "different quantifier types have different scope-taking abilities". The theory should therefore support lexical entries for quantifiers that take scope differently and compositionally in relation to each other. The present paper describes such a theory.

1.2 Why continuations

Our theory of quantifier scope is based on *continuation semantics*, which emerged [5, 6] as a compelling alternative to traditional approaches to quantification – Montague's proper treatment, Quantifier Raising (QR), type-shifting (surveyed by Barker [5]) – as well as to the Minimalism views (surveyed by Szabolcsi [2]; she also extensively discusses QR and its empirical inadequacy). Continuation semantics is compelling because it can interpret quantificational NPs (QNPs) compositionally in situ, without type-shifting, Cooper storage, or building any structures like Logical Forms beyond overt syntax. Accordingly, QNPs in subject and other positions are treated the same, QNPs and NPs are treated the same, and scope taking is semantic. Central to the approach is the hypothesis that "some linguistic expressions (in particular, QNPs) have denotations that manipulate their own continuations" [5, §1]. Although continuation semantics is only a decade old, its origin can be traced to Montague's proper treatment: "saying that NPs denote generalized quantifiers amounts to saying that NPs denote functions on their own continuations" [5, §2.2] (see also [6]). Several continuation approaches have been developed since Barker's [5], using so-called control operators [6–8] or Lambek-Grishin calculus [9].

1.3 Contributions

Like all continuation approaches, our theory features a compositional, uniform and in-situ analysis of QNPs in object, subject and other positions. Moreover, we address the following open issues.

inverse scope, scope islands and wide-scope indefinites One way to account for these phenomena is to combine control operators with metalinguistic quotation [10]. More common – see for example [11] – is using a continuation hierarchy, such as Danvy and Filinski's (D&F) hierarchy [12], which has been thoroughly investigated in the Computer Science theory. The common problem, which has not been addressed in the metalinguistic quotation and the previous D&F hierarchy approaches, is avoiding "unbound traces" – preventing denotations with unbound variables. Barker and Shan's essentially 'variable-free' semantics [13] side-steps the unbound traces problem altogether. However it relies on a different and little investigated hierarchy. The corresponding direct-style (see the next point) is unknown.

Our approach is the first to give a rigorous account of inverse scope, scope islands and wide-scope indefinites using the D&F hierarchy. We rely on types to prevent unbound traces. We formalize the pervasive intuition that a QNP is represented by a trace (QR), pronoun (Montague) or variable (Cooper

storage) that gets bound somehow. We make this intuition precise and give it logical meaning, banishing unbound traces once and for all.

direct style In Barker's continuation approach [5], every constituent's denotation explicitly receives its continuation, even though few constituents need to manipulate these continuations. Combining such *continuation-passing-style* (CPS) denotations is quite cumbersome, as we see in §2.2. Thus, we would like to avoid CPS denotations for quantifier-free constituents, in particular, for lexical entries other than quantifiers. *Direct-style* continuation semantics lets us combine continuation-manipulating denotations directly with ordinary denotations, simplifying analyses and keeping most lexical entries 'uncomplicated', which we illustrate in §2.3.

We present a version of direct-style for the D&F hierarchy. Unlike other direct-style approaches [7, 10], ours uses the ordinary λ-calculus and denotational semantics rather than operational semantics and a calculus with control operators. Our treatment of inverse scope relies on the properties of the D&F hierarchy extensively, as detailed in §4.

source of quantifier ambiguity It is common to explain quantifier ambiguity by the nondeterminism of semantic composition rules [5, 6]. One syntactic formation operation may correspond to several semantic composition functions, or the analysis may include operators like 'lift' or 'wrap' that may be freely applied to any denotation.

In contrast, our semantic composition rules are all deterministic. Although we extensively rely on schematic rules to ease notation and emphasize commonality, how these schemas are instantiated is determined unambiguously by types. Furthermore, our analysis has no optional or freely applicable rules or semantic combinators. Each syntactic formation operation maps to a unique semantic composition operation, and *vice versa*: each operation on denotations has a syntactic counterpart. This one-to-one correspondence between surface syntax and semantic composition underlies our entire approach – which is thus *directly compositional*. (See [5, §6] for the discussion of compositionality and how nondeterminism in semantic composition rules constitutes a threat.)

The source of quantifier ambiguity in our approach is solely in the lexical entries for the quantifier words rather than in the rules of syntactic formation or semantic composition. Different lexical entries for the same quantifier word have denotations corresponding to different levels of the continuation hierarchy, thus having different *strength*, or ability to scope over wider contexts.[3]

One advantage of our approach is better control over overgeneration: when only lexical entries are ambiguous, it is easier to see all available denotations and hence assure against overgeneration.

[3] The different lexical entries for the same quantifier have a regular structure. In fact, all higher-strength quantifier entries are mechanically derived from the entry for the lowest-strength quantifier, as shown in Figures 12 and 13. The number of lexical entries, that is, the assignment of the levels of strength to a quantifier is determined from empirical data.

To summarize: our contribution is a directly compositional analysis of quantifier ambiguity, scope islands, inverse linking and wide-scope indefinites in the D&F continuation hierarchy, in direct style, without risking unbound traces, and using deterministic semantic composition rules. We analyze QNP in situ and compositionally, relying on no structure beyond the overt syntax. All nondeterminism is in the choice of lexical entries for quantifier words. The presentation uses the familiar denotational semantics.

1.4 The structure of the paper

The warm-up §2 gradually introduces continuation semantics on a small fragment and explains our notation and terminology. §2.3 presents the direct-style continuation semantics as an economical CPS-on-demand. We treat bound variables rigorously in §3, with type annotations to infer variable names and to prevent unbound variables in final denotations. §4 presents the continuation hierarchy and uses it to analyze quantifier ambiguity. The corresponding direct-style, or CPS hierarchy on-demand, is described in §4.2. Scope islands, wide-scope indefinites and briefly inverse linking are the subject of §5.

For illustrations we use a small fragment of English with context-free syntax and extensional semantics, extending and refining the fragment throughout the paper. Figure 1 shows the relationship between the fragments, illustrating parallel development in CPS and direct style.

The continuation hierarchy of quantifier scope described in the paper has been implemented. The complete Haskell code is available online at `http://okmij.org/ftp/gengo/QuanCPS.hs`. The file implements the fragment of the paper in the spirit of the Penn Lambda Calculator [14], letting the user write parse trees and determine their denotations. We have used our semantic calculator for all the examples in the paper.

2 Warm-up: the proper continuation treatment of quantifiers

In this warm-up section, we recall Barker's continuation semantics [5] and summarize it in our notation. Alongside, we also introduce Barker and Shan's continuation semantics [7, 15] in direct style, which avoids pervasive type lifting of lexical entries. We use the simplicity of the examples to introduce notation and calculi to be used in further sections.

2.1 Direct semantics

Like Barker [5], we start with a simple, quantifier-free fragment, with context-free syntax and extensional semantics. The language of denotations is a plain higher-order language, Figure 2 with the obvious model-theoretical interpretation. The language has base types e and t and function types, for example $(e(et))$. We will

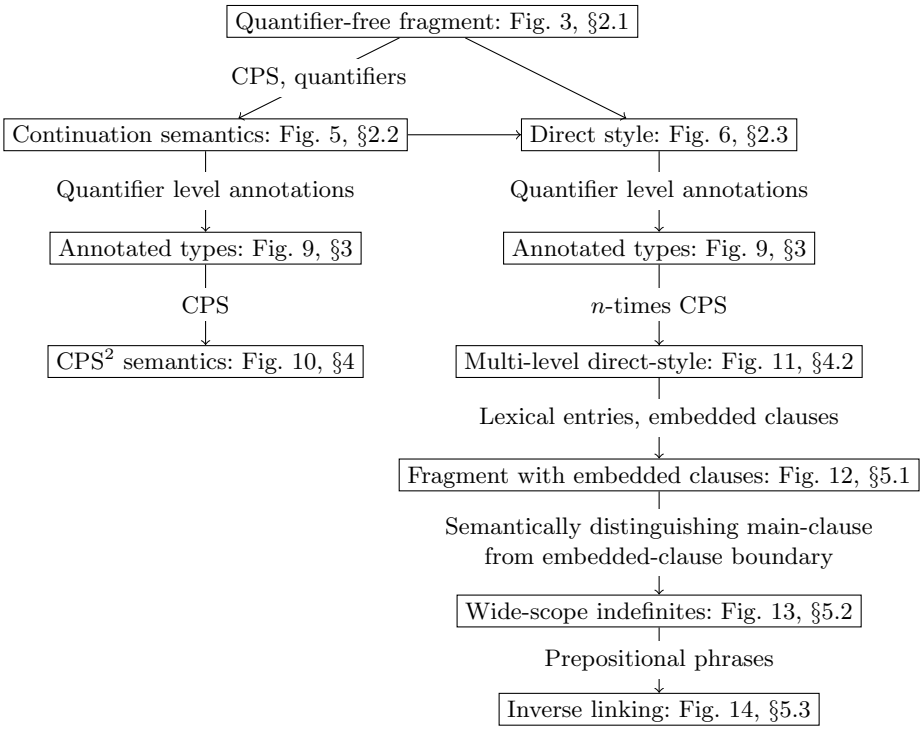

Fig. 1. Relationship between the fragments used in the paper

Base types $\upsilon ::= e \mid t$
Types $\sigma ::= \upsilon \mid (\sigma\sigma)$

Constants $c ::= \wedge \mid \vee \mid \Rightarrow \mid \neg \mid \textbf{john} \mid \textbf{mary} \mid \textbf{see} \mid \ldots$
Expressions $d ::= c \mid d\,\textbf{.}\,d$

Fig. 2. The language $\mathcal{D}$ of denotations

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
M $\rightarrow$ S .	t	$(\!\|[\![\mathsf{S}]\!]\|\!)$
S $\rightarrow$ NP VP	t	$[\![\mathsf{NP}]\!]^{<}[\![\mathsf{VP}]\!]$
VP $\rightarrow$ Vt NP	et	$[\![\mathsf{Vt}]\!]^{>}[\![\mathsf{NP}]\!]$
NP $\rightarrow$ John	e	**john**
NP $\rightarrow$ Mary	e	**mary**
VP $\rightarrow$ left	et	**leave**
Vt $\rightarrow$ saw	$e(et)$	**see**

Fig. 3. Syntax and direct semantics for a small quantifier-free fragment

often omit outer parentheses. Expressions (denoted by 'non-terminal' d) comprise constants (denoted by c) and applications $d_1 \centerdot d_2$, which are left associative: $d_1 \centerdot d_2 \centerdot d_3$ stands for $(d_1 \centerdot d_2) \centerdot d_3$. Constants are logical constants (negation, etc) and domain constants (such as **john**). Logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication) are constants of the type $t(tt)$, whose applications are written in infix, for example, $d_1 \wedge d_2$.

The syntax and semantics for our fragment is given in Figure 3. The syntax formation operation Merge corresponds to what we call forward application $^{>}$ or backward application $^{<}$ in semantics, which are defined in (8) below. [4] The notation $d_1{}^{>}d_2$ says nothing at all whether d_1 takes scope over d_2. The category M stands for the complete (matrix) sentence, terminated by the period. The corresponding semantic operation is $(\!|\cdot|\!)$. For now, these semantic composition operations are defined as follows:

$$
\begin{aligned}
d_1{}^{>}d_2 &\stackrel{\text{def}}{=} d_1 \centerdot d_2 \\
d_1{}^{<}d_2 &\stackrel{\text{def}}{=} d_2 \centerdot d_1 \\
(\!|e|\!) &\stackrel{\text{def}}{=} e
\end{aligned}
\tag{8}
$$

We extend these definitions in §2.2 when we add quantifiers, and we extend the definition of $(\!|\cdot|\!)$ a few more times. It will become clear then that the latter semantic operation is not vacuous at all. Finally, §5.1 will make it clear that $(\!|\cdot|\!)$ plays the role of the delimiter of the quantifier scope.

Figure 3 and the similar figures in the following sections demonstrate that each syntactic formation operation maps to a semantic composition operation and *vice versa*: each operation on denotations is reflected in syntax. This one-to-one syntax-semantic composition correspondence underlies our entire approach. We easily determine the denotation of a sample sentence

(9) [M [S [NP John] [VP [Vt saw] [NP Mary]]].]

[4] In our simple context-free syntax, the choice of forward or backward application is determined by the semantic types. If we used combinatorial categorial grammar (CCG), the choice of the application is evident from the categories of the nodes being combined.

Types	$\tau ::= \sigma \mid \tau \to \tau$	
Variables	x, y, z, v, f, k	
Expressions	$m ::= d \mid x \mid \lambda x.\, m \mid m\ m$	
Reductions $m \rightsquigarrow m'$	$(\lambda x.\, m)m' \rightsquigarrow m\,\{x \mapsto m'\}$	(β)

Fig. 4. Simply-typed λ-calculus, the language $\mathcal{L}$. (Base types σ and constants d are introduced in Figure 2.)

to be **see . mary . john**.

2.2 CPS semantics

We now review continuation semantics, which lets us add quantifiers to our fragment. Barker [5] has argued that the denotations of quantified phrases need access to their context. Here is a simple illustration. Suppose we had a magic domain constant **everyone** as the denotation of everyone. We could write the meaning of [M [S John [VP saw [NP everyone]]].] as $(\!|$**see . everyone . john**$|\!)$, whose model-theoretical interpretation must be the same as that of the logical formula $\forall x.$ **see .** x **. john**. Removing **everyone** from $(\!|$**see . everyone . john**$|\!)$ leaves the "term with a hole" $(\!|$**see .** $[]$ **. john**$|\!)$ – the *context* of **everyone** in the original term. We intuit that **everyone** manages to grab its context, up to the enclosing $(\!|\cdot|\!)$, and quantify over it.

To give each term the ability to grab its context, we write the terms in a *continuation-passing style* (CPS), whereupon each expression receives as an argument its context represented as a function, or *continuation*. Before we can write any CPS term, we have to resolve a small problem. To represent contexts we have to be able to build functions – an operation our language of denotations $\mathcal{D}$ (Figure 2) does not support. Therefore, we "inject" $\mathcal{D}$ into the full λ-calculus, with λ-abstractions. This calculus, or language $\mathcal{L}$, is presented in Figure 4.

The expressions of the language $\mathcal{D}$ (Figure 2) are all constants of the λ-calculus $\mathcal{L}$; the types of $\mathcal{D}$ are all base types of $\mathcal{L}$. In this sense, $\mathcal{D}$ is embedded in $\mathcal{L}$. The language $\mathcal{L}$ has its own function types, written with an arrow $\to$. Distinguishing two kinds of function types makes the continuation argument stand out in CPS terms as well as types. We exploit this distinction in §2.3.

We take $\to$ to be right associative and hence we write $t \to (t \to t)$ as $t \to t \to t$. Besides the constants, $\mathcal{L}$ has variables, abstractions and applications. The application is again left associative, with $m_1 m_2 m_3$ standing for $(m_1 m_2) m_3$.

$\mathcal{L}$ is the full λ-calculus and has reductions, $m \rightsquigarrow m'$. An expression is in normal form if no reduction applies to it or any of its sub-expressions. The notation $m\,\{x \mapsto m'\}$ in the β-reduction rule stands for the capture-avoiding substitution of m' for x in m. A unique normal form always exists and can be reached by any sequence of reductions; in other words, $\mathcal{L}$ is strongly normalizing.

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
M $\rightarrow$ S .	t	$(\!\|[\![\mathsf{S}]\!]\|\!)$
S $\rightarrow$ NP VP	$(t \to t) \to t$	$[\![\mathsf{NP}]\!]^{<}[\![\mathsf{VP}]\!]$
VP $\rightarrow$ Vt NP	$((et) \to t) \to t$	$[\![\mathsf{Vt}]\!]^{>}[\![\mathsf{NP}]\!]$
NP $\rightarrow$ John	$(e \to t) \to t$	$\lambda k.\, k\ \mathbf{john}$
NP $\rightarrow$ Mary	$(e \to t) \to t$	$\lambda k.\, k\ \mathbf{mary}$
VP $\rightarrow$ left	$((et) \to t) \to t$	$\lambda k.\, k\ \mathbf{leave}$
Vt $\rightarrow$ saw	$((e(et)) \to t) \to t$	$\lambda k.\, k\ \mathbf{see}$
NP $\rightarrow$ everyone	$(e \to t) \to t$	$\lambda k.\, \forall x.\, k\ x$
NP $\rightarrow$ someone	$(e \to t) \to t$	$\lambda k.\, \exists x.\, k\ x$

Fig. 5. Syntax and continuation semantics for the small fragment

We are set to write CPS denotations for our fragment. Constants like **john** have little to do but to "plug themselves" into their context: $\lambda k.\, k\ \mathbf{john}$.[5] Here k represents the context of **john** within the whole sentence denotation. The whole denotation must be of the type t; hence k has the type $e \to t$ and the type of the CPS form of **john** is $(e \to t) \to t$. With the CPS denotations, our fragment now reads as in Figure 5. The semantic composition operators are now defined as follows.

$$
\begin{aligned}
m_1{}^{>}m_2 &\stackrel{\text{def}}{=} \lambda k.\, m_1(\lambda f.\, m_2(\lambda x.\, k(f\ \centerdot\ x))) \\
m_1{}^{<}m_2 &\stackrel{\text{def}}{=} \lambda k.\, m_1(\lambda x.\, m_2(\lambda f.\, k(f\ \centerdot\ x))) \\
(\!|m|\!) &\stackrel{\text{def}}{=} m(\lambda v.\, v)
\end{aligned}
\tag{10}
$$

The CPS form of $m_1{}^{>}m_2$ is $\lambda k.\, m_1(\lambda f.\, m_2(\lambda x.\, k\ (f\ \centerdot\ x)))$: it fills its context k with $f\ \centerdot\ x$, where f is what m_1 fills its context with, and x is what m_2 fills its context with.

Using Figure 5 to compute the denotation of the sample sentence (9) gives us:

$$
\begin{aligned}
&(11) \quad [\![[_{\mathsf{M}}\ [_{\mathsf{S}}\ [_{\mathsf{NP}}\ \mathsf{John}]\ [_{\mathsf{VP}}\ [_{\mathsf{Vt}}\ \mathsf{saw}]\ [_{\mathsf{NP}}\ \mathsf{Mary}]]]\,.]]\!] \\
&\quad = (\lambda k_0.\, (\lambda k.\, k\ \mathbf{john})(\lambda x. \\
&\qquad (\lambda k_1.\, (\lambda k.\, k\ \mathbf{see})(\lambda f'.\, (\lambda k.\, k\ \mathbf{mary})(\lambda x'.\, k_1\ (f'\ \centerdot\ x'))))\\
&\qquad (\lambda f.\, k_0\ (f\ \centerdot\ x))))\\
&\qquad (\lambda v.\, v) \\
&\quad \rightsquigarrow (\lambda k_0.\, (\lambda k.\, k\ \mathbf{john})(\lambda x. \\
&\qquad (\lambda k_1.\, (\lambda k.\, k\ \mathbf{mary})(\lambda x'.\, k_1\ (\mathbf{see}\ \centerdot\ x'))) \\
&\qquad (\lambda f.\, k_0\ (f\ \centerdot\ x)))) \\
&\qquad (\lambda v.\, v)
\end{aligned}
$$

[5] When a context is represented by a continuation function k, filling the hole in the context with a term e – or, plugging e into the context – is represented by the application $k\ e$.

$$
\begin{aligned}
&\rightsquigarrow (\lambda k_0.\,(\lambda k.\,k\ \mathbf{john})(\lambda x.\\
&\qquad (\lambda k_1.\,k_1\ (\mathbf{see}\,.\,\mathbf{mary}))\\
&\qquad (\lambda f.\,k_0\ (f\,.\,x))))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,(\lambda k.\,k\ \mathbf{john})(\lambda x.\,(k_0\ ((\mathbf{see}\,.\,\mathbf{mary})\,.\,x))))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,(k_0\ ((\mathbf{see}\,.\,\mathbf{mary})\,.\,\mathbf{john})))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow ((\mathbf{see}\,.\,\mathbf{mary})\,.\,\mathbf{john})
\end{aligned}
$$

The β-reductions lead to the same expression $((\mathbf{see}\,.\,\mathbf{mary})\,.\,\mathbf{john})$ as in §2.1. The argument k_1 was the continuation of $[\![\mathsf{saw\ Mary}]\!]$. The term $(\lambda k_0 \ldots)$ was the denotation of the main clause $[_{\mathsf{S}}\ \mathsf{John}\ [_{\mathsf{VP}}\ \mathsf{saw\ Mary}]]$, whose context is empty, represented by $\lambda v.\,v$. (If the clause were an embedded one, its context would not have been empty. We discuss embedded clauses in §5.1.)

Figure 5 contains two extra rows, not present in Figure 3: The CPS semantics lets us express QNPs. The denotation of everyone, $\lambda k.\,\forall x.\,k\ x$, is what we have informally argued at the beginning of §2.2 the denotation of everyone should be: the quantifier grabs its continuation k and quantifies over it. The denotation is a bit sloppy since we have not yet introduced quantifiers in any of our languages, $\mathcal{D}$ or $\mathcal{L}$. Such an informal style, appealing to predicate logic, is very common. For now, we go along; we come back to this point in §3, arguing that it pays to be formal. Let us see how quantification works:

(12) $[\![[_{\mathsf{M}}\ [_{\mathsf{S}}\ [_{\mathsf{NP}}\ \mathsf{John}]\ [_{\mathsf{VP}}\ [_{\mathsf{Vt}}\ \mathsf{saw}]\ [_{\mathsf{NP}}\ \mathsf{everyone}]]].]\!]$

$$
\begin{aligned}
&= (\lambda k_0.\,(\lambda k.\,k\ \mathbf{john})(\lambda x.\\
&\qquad (\lambda k_1.\,(\lambda k.\,k\ \mathbf{see})(\lambda f'.\,(\lambda k.\,\forall x''.\,k\ x'')(\lambda x'.\,k_1\ (f'\,.\,x'))))\\
&\qquad (\lambda f.\,k_0\ (f\,.\,x))))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,(\lambda k_1.\,(\lambda k.\,\forall x''.\,k\ x'')(\lambda x'.\,k_1\ (\mathbf{see}\,.\,x'))))\\
&\qquad (\lambda f.\,k_0\ (f\,.\,\mathbf{john})))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,(\lambda k_1.\,\forall x''.\,k_1\ (\mathbf{see}\,.\,x''))\\
&\qquad (\lambda f.\,k_0\ (f\,.\,\mathbf{john})))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,\forall x''.\,k_0\ (\mathbf{see}\,.\,x'')\,.\,\mathbf{john})\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow \forall x''.\,(\mathbf{see}\,.\,x'')\,.\,\mathbf{john}
\end{aligned}
$$

The sample sentence "John saw everyone" had the quantifier in the object position, and yet we, unlike Montague, did not have to do anything special to accommodate it. In fact, comparing (11) against (12) shows that everyone is treated just like Mary. The β-reductions accumulate the context captured by the quantifier until it eventually becomes the full sentence context.

A quantifier in the subject position, unlike with QR, is treated just like a quantifier in the object position:

(13) $[\![[_{\mathsf{M}}\ [_{\mathsf{S}}\ [_{\mathsf{NP}}\ \mathsf{Someone}]\ [_{\mathsf{VP}}\ [_{\mathsf{Vt}}\ \mathsf{saw}]\ [_{\mathsf{NP}}\ \mathsf{everyone}]]].]]\!]$

$$
\begin{aligned}
&= (\lambda k_0.\,(\lambda k.\,\exists y.\,k\ y)(\lambda x.\\
&\qquad (\lambda k_1.\,(\lambda k.\,k\ \mathbf{see})(\lambda f'.\,(\lambda k.\,\forall x''.\,k\ x'')(\lambda x'.\,k_1\ (f'\centerdot x'))))\\
&\qquad (\lambda f.\,k_0\ (f\centerdot x))))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,(\lambda k.\,\exists y.\,k\ y)(\lambda x.\\
&\qquad (\lambda k_1.\,\forall x''.\,k_1\ (\mathbf{see}\centerdot x''))\\
&\qquad (\lambda f.\,k_0\ (f\centerdot x))))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,\exists y.\,(\lambda k_1.\,\forall x''.\,k_1\ (\mathbf{see}\centerdot x''))(\lambda f.\,k_0\ (f\centerdot y)))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow (\lambda k_0.\,\exists y.\,\forall x''.\,k_0((\mathbf{see}\centerdot x'')\centerdot y))\\
&\quad (\lambda v.\,v)\\
&\rightsquigarrow \exists y.\,\forall x''.\,(\mathbf{see}\centerdot x'')\centerdot y
\end{aligned}
$$

Thus, continuation semantics can treat QNPs in any syntactic position with no type-shifting and no surgery on the syntactic derivation. The resulting denotation for "Someone saw everyone" is the linear-scope reading. Deriving the inverse-scope reading is the subject of §4.

2.3 Direct-style continuation semantics

This section describes a "direct style" advocated by Barker and Shan [7, 11, 15]. Its great appeal is in simple, non-CPS denotations for quantifier-free phrases. In particular, lexical entries other than quantifiers keep their straightforward mapping to domain constants, like the mapping in Figure 3. Our presentation of direct style is different from that of Shan [7]: we use the ordinary λ-calculus and the denotational semantics, without introducing operational semantics and so-called control operators (although the informed reader will readily recognize these operators in our presentation). We introduce direct style as 'CPS on-demand'.

We start with an observation about CPS denotations:

$$
\begin{aligned}
[\![\mathsf{John}]\!] &= \lambda k.\,k\ \mathbf{john}\\
[\![\mathsf{saw\ Mary}]\!] &= \lambda k.\,k\ (\mathbf{see}\centerdot\mathbf{mary})
\end{aligned}
$$

In general, the CPS denotation of a quantifier-free term can be built by first determining the denotation according to the non-CPS rules (8), then wrapping $\lambda k.\,k\ (\cdot)$ around the result.

This observation gives us the idea to merge quantifier-free and CPS semantics; see Figure 6. If denotations are quantifier-free – that is, if their types have no arrows – we use the non-CPS composition rules (8), which constitute the first case in (14) and (15) below. For CPS denotations, we use the CPS composition

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
$\mathsf{M} \to \mathsf{S}\ .$	t	$(\!\vert [\![\mathsf{S}]\!] \vert\!)$
$\mathsf{S} \to \mathsf{NP\ VP}$	t or $(t \to t) \to t$	$[\![\mathsf{NP}]\!]\ ^{<}\ [\![\mathsf{VP}]\!]$
$\mathsf{VP} \to \mathsf{Vt\ NP}$	et or $((et) \to t) \to t$	$[\![\mathsf{Vt}]\!]\ ^{>}\ [\![\mathsf{NP}]\!]$
$\mathsf{NP} \to \mathsf{John}$	e	**john**
$\mathsf{NP} \to \mathsf{Mary}$	e	**mary**
$\mathsf{VP} \to \mathsf{left}$	et	**leave**
$\mathsf{Vt} \to \mathsf{saw}$	$e(et)$	**see**
$\mathsf{NP} \to \mathsf{everyone}$	$(e \to t) \to t$	$\lambda k.\, \forall x.k\ x$
$\mathsf{NP} \to \mathsf{someone}$	$(e \to t) \to t$	$\lambda k.\, \exists x.k\ x$

Fig. 6. Syntax and direct-style continuation semantics for the small fragment: the merger of Figures 3 and 5. Lexical entries other than the quantifiers keep the simple denotations from Figure 3.

rules (10), written as the last case in (14) and (15). When composing CPS and non-CPS denotations, we implicitly promote the latter into CPS by wrapping them in $\lambda k.\, k\ (\cdot)$. The two middle cases of (14) and (15) show the result of that promotion after simplification (β-reductions). Thus the composition rules $^{>}$ and $^{<}$ become schematic with four cases. Likewise, $(\!\vert \cdot \vert\!)$ becomes schematic with two cases, shown in (16). We stress the absence of any nondeterminism: which of the four composition rules to apply is uniquely determined by the types of the denotations being combined.

$$(14)\quad m_1{}^{>}m_2 \stackrel{\text{def}}{=} \begin{cases} m_1\,.\,m_2 & \text{if } m_1 : (\sigma\sigma'),\ m_2 : \sigma \\ \lambda k.\, m_2(\lambda x.\, k(m_1\,.\,x)) & \text{if } m_1 : (\sigma\sigma'),\ m_2 : (\sigma \to t) \to t \\ \lambda k.\, m_1(\lambda f.\, k(f\,.\,m_2)) & \text{if } m_1 : ((\sigma\sigma') \to t) \to t,\ m_2 : \sigma \\ \lambda k.\, m_1(\lambda f.\, m_2(\lambda x.\, k(f\,.\,x))) & \text{if } m_1 : ((\sigma\sigma') \to t) \to t, \\ & \quad m_2 : (\sigma \to t) \to t \end{cases}$$

$$(15)\quad m_1{}^{<}m_2 \stackrel{\text{def}}{=} \begin{cases} m_2\,.\,m_1 & \text{if } m_1 : \sigma,\ m_2 : (\sigma\sigma') \\ \lambda k.\, m_2(\lambda f.\, k(f\,.\,m_1)) & \text{if } m_1 : \sigma,\ m_2 : ((\sigma\sigma') \to t) \to t \\ \lambda k.\, m_1(\lambda x.\, k(m_2\,.\,x)) & \text{if } m_1 : (\sigma \to t) \to t,\ m_2 : (\sigma\sigma') \\ \lambda k.\, m_1(\lambda x.\, m_2(\lambda f.\, k(f\,.\,x))) & \text{if } m_1 : (\sigma \to t) \to t, \\ & \quad m_2 : ((\sigma\sigma') \to t) \to t \end{cases}$$

$$(16)\qquad (\!\vert m \vert\!) \stackrel{\text{def}}{=} \begin{cases} m & \text{if } m : t \\ m(\lambda v.\, v) & \text{if } m : (t \to t) \to t \end{cases}$$

Since the sentence [$_{\mathsf{M}}$ John [$_{\mathsf{VP}}$ saw Mary].] is quantifier-free, its denotation is trivially determined as in §2.1, with no β-reductions – in marked contrast with §2.2. For [$_{\mathsf{M}}$ Someone [$_{\mathsf{VP}}$ saw Mary].], we compute $[\![$[$_{\mathsf{VP}}$ saw Mary]$]\!]$ as **see . mary**

Levels	$n, l \in \mathbb{N}$
Base types	$\upsilon ::= e \mid t$
Types	$\sigma ::= \upsilon \mid (\sigma\sigma)$
Annotated types	$\rho ::= \sigma^n$
Constants	$c ::= \wedge \mid \vee \mid \Rightarrow \mid \neg \mid \mathbf{john} \mid \mathbf{mary} \mid \mathbf{see} \mid \ldots$
Variables	n, l
Expressions	$d ::= c \mid d \centerdot d \mid n \mid \forall_n d \mid \exists_n d$

Type system for judgments $d : \rho$

$$\frac{}{n : e^{n+1}} \qquad \frac{d_1 : (\sigma_2\sigma_1)^{n_1} \quad d_2 : \sigma_2^{n_2}}{d_1 \centerdot d_2 : \sigma_1^{\max(n_1,n_2)}} \qquad \frac{d : t^{n+1}}{\forall_n d : t^n} \qquad \frac{d : t^{n+1}}{\exists_n d : t^n}$$

Fig. 7. The language $\mathcal{D}_{\mathcal{Q}}$ of denotations

of the type (et) by the simple rules of (8). The denotation of $\mathsf{someone}$ has the type $(e \to t) \to t$, which is a CPS type: it has arrows. The types tell us to use the third case of (15) to combine $[\![\mathsf{someone}]\!]$ with $[\![[\![\mathsf{VP\ saw\ Mary}]]\!]]$. We obtain the final result $\exists y.\, \mathbf{see} \centerdot \mathbf{mary} \centerdot y$ after applying the second case of (16).

Direct style thus keeps quantifier-free lexical entries 'unlifted' and removes the tedium of the CPS semantics. Such CPS-on-demand, or *selective* CPS, has been used to implement delimited control in Scala [16].

3 The nature of quantification

Before we advance to the main topic, scope and ambiguity, we take a hard look at logical quantification. So far, we have used quantified logical formulas like $\forall x.\, \mathbf{see} \centerdot x \centerdot \mathbf{john}$ without formally introducing quantifiers. The informality, however attractive, makes it hard to specify how to correctly use a logical quantifier to obtain a well-formed closed formula. For example, QR approaches may produce a denotation with an unbound trace, which must then be somehow fixed or avoided. A proper theory should not let sentence denotations with unbound variables arise in the first place.

We go back to the language $\mathcal{D}$, Figure 2, and extend it with standard first-order quantifiers. The result is the language $\mathcal{D}_{\mathcal{Q}}$ in Figure 7.

We added variables, which are natural numbers, and two expression forms $\forall_n d$ and $\exists_n d$ to quantify over the variable n. Their model-theoretical semantics is standard, relying on the variable assignment ϕ, which maps variables to entities. Then $\forall_n d$ is true for the assignment ϕ iff d is true for every assignment that differs from ϕ only in the mapping of the variable n.

Figure 7 also extends the type system, with annotated types ρ and judgments $d : \rho$ of d having the annotated type ρ. Expression types σ are annotated with the upper bound on the variable names that may occur in the expression. For

$$\frac{m_1 : (\sigma_2\sigma_1)^{n_1} \quad m_2 : \sigma_2^{n_2}}{m_1{}^{>}m_2 : \sigma_1^{\max(n_1,n_2)}} \qquad \frac{m_1 : (\sigma_2\sigma_1)^{n_1} \quad m_2 : (\sigma_2^{n_2} \to t^{l_1}) \to t^{l_2}}{m_1{}^{>}m_2 : (\sigma_1^{\max(n_1,n_2)} \to t^{l_1}) \to t^{l_2}}$$

$$\frac{m_1 : ((\sigma_2\sigma_1)^{n_1} \to t^{l_1}) \to t^{l_2} \quad m_2 : \sigma_2^{n_2}}{m_1{}^{>}m_2 : (\sigma_1^{\max(n_1,n_2)} \to t^{l_1}) \to t^{l_2}}$$

$$\frac{m_1 : ((\sigma_2\sigma_1)^{n_1} \to t^{l_1}) \to t^{l_2} \quad m_2 : (\sigma_2^{n_2} \to t^{l_3}) \to t^{l_1}}{m_1{}^{>}m_2 : (\sigma_1^{\max(n_1,n_2)} \to t^{l_3}) \to t^{l_2}} \qquad \frac{m : t^0}{(\!|m|\!) : t^0} \qquad \frac{m : (t^n \to t^n) \to t^0}{(\!|m|\!) : t^0}$$

Fig. 8. Typing rules for $^{>}$ in (14) ($^{<}$ is analogous) and for $(\!|\cdot|\!)$ in (16).

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
...		
NP $\to$ everyone	$(e^{n+1} \to t^{n+1}) \to t^n$	$\lambda k.\, \forall_n(k\ n)$
NP $\to$ someone	$(e^{n+1} \to t^{n+1}) \to t^n$	$\lambda k.\, \exists_n(k\ n)$

Fig. 9. Precise denotations of quantifiers and their annotated types. The rest of the fragment remains the same; see Figure 5 or 6.

example, $d : \sigma^1$ means that d *may* have (several) occurrences of the variable 0; $d : \sigma^2$ means d may contain the variables 0 and 1. Our variables are de Bruijn *levels*. An expression d of the type σ^0 is a closed expression. We will often omit the type annotation (superscript) 0 – hence $\mathcal{D}$ can be regarded as the variable-free fragment of $\mathcal{D}_\mathcal{Q}$.

The language $\mathcal{L}$ will now use the expressions of $\mathcal{D}_\mathcal{Q}$ as constants, and annotated types ρ as base types. Although the semantic composition functions in (14), (15) and (16) remain the same, their typing becomes more precise, as shown in Figure 8. (Recall $(\!|\cdot|\!)$ is the semantic composition function that corresponds to the clause boundary, which we will discuss in detail in §5.1.) As usual, the typing rules are schematic: m_1 and m_2 stand for arbitrary expressions of $\mathcal{L}$, σ_1 and σ_2 stand for arbitrary $\mathcal{D}_\mathcal{Q}$ types, and n_1, n_2, l_1, l_2, etc. are arbitrary levels. The choice n or l for the name of level metavariables has no significance beyond notational convenience. The English fragments in Figures 5 and 6 remain practically the same; the quantifier words now receive precisely defined rather than informal denotations, and precise semantic types; see Figure 9.

Figure 9 assigns denotations and types to everyone and someone that are schematic in n. That is, there is an instance of the denotation for each natural number n. One may worry about choosing the right n and possible ambiguities. The worries are unfounded. As we demonstrate below, the requirement that the whole sentence denotation be closed (that is, have the type t^0) uniquely determines the choice of n in the denotation schemas for the quantifier words. The choice of variable names n is hence type-directed and deterministic. As an example, we show the typing derivation for "Someone saw everyone", which we

explain below.

$$\dfrac{\dfrac{[\![\mathsf{someone}]\!] : (e^1 \to t^1) \to t^0 \qquad \dfrac{[\![\mathsf{see}]\!] : e(et)^0 \quad [\![\mathsf{everyone}]\!] : (e^2 \to t^2) \to t^1}{\mathbf{see}^{>}(\lambda k.\, \forall_1(k1)) : ((et)^2 \to t^2) \to t^1}}{(\lambda k.\, \exists_0(k0))^{<}(\mathbf{see}^{>}\lambda k.\, \forall_1(k1)) : (t^2 \to t^2) \to t^0}}{(\!|(\lambda k.\, \exists_0(k0))^{<}(\mathbf{see}^{>}\lambda k.\, \forall_1(k1))|\!) : t^0}$$

The resulting denotation β-reduces to $\exists_0\forall_1\mathbf{see}\centerdot 1\centerdot 0$, as in §2.2. The other derivations in §2.2 and §2.3 are made rigorous similarly.

In the derivation above, the schematic denotation $[\![\mathsf{someone}]\!]$ was instantiated with $n = 0$, and the schema $[\![\mathsf{everyone}]\!]$ was instantiated with $n = 1$. It may be unclear how we have made this choice. It is a simple exercise to see that no other choice fits. Relying on the simplicity of the example, we now demonstrate the general method of choosing the variable names n appearing in schematic denotations. We repeat the derivation, this time assuming that $[\![\mathsf{someone}]\!]$ is instantiated with some variable name n and $[\![\mathsf{everyone}]\!]$ is instantiated with some name l. These so-called *schematic* or logical meta-variables n and l stand for some natural numbers that we do not know yet. As we build the derivation and fit the denotations, we discover constraints on n and l, which in the end let us determine these numbers.

$$\dfrac{\dfrac{[\![\mathsf{someone}]\!] : (e^{n+1} \to t^{n+1}) \to t^n \qquad \dfrac{[\![\mathsf{see}]\!] : e(et)^0 \quad [\![\mathsf{everyone}]\!] : (e^{l+1} \to t^{l+1}) \to t^l}{\mathbf{see}^{>}(\lambda k.\, \forall_l(k\,l)) : ((et)^{l+1} \to t^{l+1}) \to t^l}}{(\lambda k.\, \exists_n(k\,n))^{<}(\mathbf{see}^{>}\lambda k.\, \forall_l(k\,l)) : (t^{\max(n+1,l+1)} \to t^{l+1}) \to t^n \quad \text{where } n+1=l}}{(\!|(\lambda k.\, \exists_n(k\,n))^{<}(\mathbf{see}^{>}\lambda k.\, \forall_l(k\,l))|\!) : t^0 \quad \text{where } n=0,\ \max(n+1, l+1) = l+1}$$

In the last-but-one step of the derivation, we attempt to type $(\lambda k.\, \exists_n(k\ n))^{<}(\mathbf{see}^{>}\lambda k.\, \forall_l(k\ l))$ using the rule

$$\dfrac{m_1 : (\sigma_2^{n_1} \to t^{l_1}) \to t^{l_2} \quad m_2 : ((\sigma_2\sigma_1)^{n_2} \to t^{l_3}) \to t^{l_1}}{m_1{}^{<}m_2 : (\sigma_1^{\max(n_1,n_2)} \to t^{l_3}) \to t^{l_2}}.$$

This attempt only works if $n + 1 = l$, because according to the rule, the types of m_1 and m_2 must share the same name l_1. In the last step of the derivation, applying the typing rule for $(\!|\cdot|\!)$ from Figure 8 gives two other constraints: $n = 0$ and $\max(n + 1, l + 1) = l + 1$. The three constraints have a unique solution: $n = 0$, $l = 1$.

More complex sentences with more quantifiers require us to deal with more variable names n_1, n_2, n_3, etc., and more constraints on them. The overall principle remains straightforward: since typing is syntax-directed there is never a puzzle as to which typing rule to use at any stage of the derivation. At most one typing rule applies. An application of a typing rule generally imposes constraints on the levels. We collect all constraints and solve them at the end (some constraints can be solved as we go).

Accumulating and solving such constraints is a logic programming problem. Luckily, in modern functional and logic programming languages like Haskell,

Twelf or Agda, type checking propagates and solves constraints in a very similar way. If we write our denotations in, say, Haskell, the Haskell type checker automatically determines the names of schematic meta-variables and resolves schematic denotations and rules. We have indeed used the Haskell interpreter GHCi as such a 'semantic calculator', which infers types, builds derivations and instantiates schemas. Like the Penn Lambda Calculator [14], the Haskell interpreter also reduces terms. We can enter any syntactic derivation at the interpreter prompt and see its inferred type and its normal-form denotation.

The choice of variable names, dictated by the requirement that sentence denotations be closed, in turn describes quantifier scopes, as we shall see next.

4 The inverse-scope problem

If we compute the denotation of $[_{\mathsf{M}}\ \mathsf{Someone\ VP.}]$ by the rules of §2.2, we obtain

$$\begin{aligned}(17)\qquad [\![\mathsf{Someone\ VP.}]\!] &= (\lambda k_0.\,(\lambda k.\,\exists y.ky)(\lambda x.\,[\![\mathsf{VP}]\!]\,(\lambda f.\,k_0(f\centerdot x))))\\ &\qquad\quad (\lambda v.\,v)\\ &\rightsquigarrow \exists y.\ [\![\mathsf{VP}]\!]\,(\lambda f.\,(f\centerdot y))\end{aligned}$$

No matter what VP is, the existential always scopes over it. Thus, we invariably get the linear-scope reading for the sentence. Obtaining the inverse-scope reading is the problem. One suggested solution [5, 6] is to introduce nondeterminism into semantic composition rules. We do not find that approach attractive because of over-generation: we may end up with a great number of denotations, not all of which correspond to available readings. Explaining different scope-taking abilities of existentials and universals (see §5) also becomes very difficult.

Our solution to inverse scope is the *continuation hierarchy* [12]. Like Russian dolls, contexts nest. Plugging a term into a context gives a bigger term, which can be plugged into another, wider context, and so on. This hierarchy of contexts is reflected in the continuation hierarchy. Quantifiers gain access not only to their immediate context but also to a higher-up context, and may hence quantify over outer contexts. We build the hierarchy from the CPS denotations of §2.2, to be called CPS^1 denotations (with the annotated types of §3). We introduce the corresponding direct style of the hierarchy in §4.2.

Before we begin, let us quickly skip ahead and peek at the final result, to see the diffference that the continuation hierarchy makes. Eq. (17) will look somewhat like

$$\begin{aligned}(17')\qquad [\![\mathsf{Someone\ VP.}]\!] &= (\lambda k_0.\,(\lambda k_1.\,\lambda k_2.\,k_1\ y\ (\lambda v.\,k_2(\exists y.v)))\\ &\qquad\qquad (\lambda x.\,[\![\mathsf{VP}]\!]\,(\lambda f.\,k_0\ (f\centerdot x))))\\ &\qquad\quad (\lambda v.\,\lambda k_2.\,k_2\ v)(\lambda v.\,v)\\ &\rightsquigarrow [\![\mathsf{VP}]\!]\,(\lambda f.\,\lambda k_2.\,k_2(f\centerdot y))(\lambda v.\,(\exists y.v))\end{aligned}$$

(see Eq. (25) for the complete example). VP will now have a chance to introduce a quantifier to scope over $\exists y.\cdot$.

We build the hierarchy by iterating the CPS transformation. An expression may be re-written in CPS multiple times. Each re-writing adds another continuation representing a higher (outer) context [12]. Let us take an example. A term **john** written in CPS takes the continuation argument representing the term's context, and plugs itself into that context: $\lambda k.\, k$ **john**. Mechanically applying to it the rules of transforming terms into CPS [12] gives $\lambda k_1.\, \lambda k_2.\, (k_1$ **john**$)\ k_2$. This CPS^2 term receives two continuations and plugs **john** into the inner one, obtaining the CPS^1 term k_1 **john** that computes the result to be plugged into the outer context k_2. We may diagram the CPS^1 term $\lambda k_1.\, k_1$ **john** as $[_{k_1} \ldots [\mathbf{john}] \ldots]$, that is, **john** filling in the hole in a context represented by k_1. Likewise we diagram the CPS^2 term $\lambda k_1.\, \lambda k_2.\, (k_1$ **john**$)\ k_2$ as $[_{k_2} \ldots [_{k_1} \ldots [\mathbf{john}] \ldots] \ldots]$. In the CPS^2 case, if k_2 represents the outer context, the application $k_2\ e$ represents plugging e into that context. If k_1 is an inner context, $k_1\ e\ k_2$ corresponds to plugging e into it and the result into an outer context k_2. We shall see soon that types make it clear which context, outer or inner, a continuation represents and what needs to be plugged into what.

The CPS^2 term $\lambda k_1.\, \lambda k_2.\, k_1$ **john** k_2 is however extensionally equivalent to the CPS^1 term $\lambda k.\, k$ **john** we started with. In general, if a term uses its continuation 'trivially',[6] further CPS transformations leave the term intact. Thus, after quantifier-free lexical entries are converted once into CPS, they can be used as they are at any level of the CPS hierarchy.

Although the CPS^2 term of **john** is same as the CPS^1 term, the types differ. The CPS^1 type is $(e \to t^n) \to t^n$, telling us that **john** receives a context to be plugged with a term of the type e giving a term of the type t^n. The CPS^2-term receives another continuation k_2, representing the outer context $t^n \to t^{l_1}$. Thus the type of $\lambda k_1.\, \lambda k_2.\, k_1$ **john** k_2 is $(e \to ((t^n \to t^{l_1}) \to t^{l_2})) \to ((t^n \to t^{l_1}) \to t^{l_2})$. This type is schematic, written with schematic meta-variables n, l_1 and l_2 standing for some variable names to be determined when building a derivation, as described in §3.

In general, types in the CPS hierarchy have a regular structure and can be described uniformly. The key observation is recurrence of the pattern $(t^n \to t^{l_1}) \to t^{l_2}$ that can be represented by its sequence of annotations n, l_1, l_2. Therefore, we introduce the notation

$$
\begin{aligned}
\{n\} &= t^n \\
\{n l_1 l_2\} &= (t^n \to \{l_1\}) \to \{l_2\} \\
\{n l_1 l_2 l_3 l_4 l_5 l_6\} &= (t^n \to \{l_1 l_2 l_3\}) \to \{l_4 l_5 l_6\} \\
&\vdots
\end{aligned}
\tag{18}
$$

where all ns and ls are schematic meta-variables. Since these sequences can become very long, we use Greek letters α, β, γ to each stand for a schematic *sequence* of variable names. All occurrences of the same Greek letter bearing the same superscripts and subscripts refer to the same sequence. We will state

[6] We say that a term uses its continuation argument k trivially if k is used exactly once in the term, and each application in the term is the entire body of a λ-abstraction.

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
M $\to$ S .	t^0	$(\!\|[\![\mathsf{S}]\!]\|\!)$
S $\to$ NP VP	$(t^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{NP}]\!]\ ^{<}\ [\![\mathsf{VP}]\!]$
VP $\to$ Vt NP	$((et)^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{Vt}]\!]\ ^{>}\ [\![\mathsf{NP}]\!]$
NP $\to$ John	$(e \to \{\alpha\}) \to \{\alpha\}$	$\lambda k.\, k$ **john**
NP $\to$ Mary	$(e \to \{\alpha\}) \to \{\alpha\}$	$\lambda k.\, k$ **mary**
VP $\to$ left	$((et) \to \{\alpha\}) \to \{\alpha\}$	$\lambda k.\, k$ **leave**
Vt $\to$ saw	$((e(et)) \to \{\alpha\}) \to \{\alpha\}$	$\lambda k.\, k$ **see**
NP $\to$ everyone$_1$	$(e^{n+1} \to \{(n+1)\gamma\}) \to \{n\gamma\}$	$\lambda k_1.\, \lambda k_2.\, k_1\ n\ (\lambda v.\, k_2(\forall_n v))$
NP $\to$ someone$_1$	$(e^{n+1} \to \{(n+1)\gamma\}) \to \{n\gamma\}$	$\lambda k_1.\, \lambda k_2.\, k_1\ n\ (\lambda v.\, k_2(\exists_n v))$
NP $\to$ everyone$_2$	$(e^{n+1} \to \{\gamma(n+1)\}) \to \{\gamma n\}$	$\lambda k_1.\, \lambda k_2.\, \forall_n(k_1\ n\ k_2)$
NP $\to$ someone$_2$	$(e^{n+1} \to \{\gamma(n+1)\}) \to \{\gamma n\}$	$\lambda k_1.\, \lambda k_2.\, \exists_n(k_1\ n\ k_2)$

Fig. 10. Syntax and the CPS^2 semantics for the small fragment. α and β are sequences of schematic meta-variables of length 3, and γ is a sequence of length 2. See the text for expressions and types of the semantic composition operators $^{>}$, $^{<}$ and $(\!|\cdot|\!)$

the length of the sequence separately or leave it implicit in the CPS level under discussion. Thus the type of $\lambda k.\, k$ **john** for any CPS level has the form $(e \to \{\alpha\}) \to \{\alpha\}$. Juxtaposed Greek letters and schematic variables signify concatenated sequences. For example, (18) is compactly written as follows.

$$\begin{aligned} \{n\} &= t^n \\ \{n\alpha\beta\} &= (t^n \to \{\alpha\}) \to \{\beta\} \end{aligned} \tag{19}$$

4.1 CPS-hierarchy semantics

The CPS^2 semantics for our language fragment is shown in Figure 10. Except for the quantifiers, the figure looks like the ordinary CPS semantics, Figure 5, with the wholesale replacement of the type t by $\{\alpha\}$. The interesting part is quantifier words. There are now two sets of them, indexed with 1 and 2: the quantifier words become polysemous, with two possible denotations. Postulating the polysemy of quantifiers is similar to generalizing the conjunction schema [17], or assuming the free indexing in LF.

The quantifiers everyone$_1$ and someone$_1$ are the quantifiers from §2.2, whose denotations are re-written in CPS. For example, the denotation of everyone from Figure 9 (which is the precise version of that from Figure 5) is $\lambda k.\, \forall_n(k\ n)$; rewriting it in CPS gives $\lambda k_1.\, \lambda k_2.\, k_1\ n\ (\lambda v.\, k_2(\forall_n v))$. It plugs the variable n into the (inner) context k_1, then plugs the result into $\forall_n[]$ and finally into the outer context k_2. Thus, everyone$_1$ quantifies over the immediate, inner context k_1, as in §2.2 above. The continuation arguments to everyone$_1$ are used trivially, so the denotation can be used as it is not only for CPS^2 but also for CPS^3 and at higher levels.

The second set of quantifiers quantify over the outer context, as their denotation says. For example, $\lambda k_1.\, \lambda k_2.\, \forall_n(k_1\ n\ k_2)$ plugs the variable n into the

inner context k_1, plugs the result into k_2 and quantifies over the final result. The inner and the outer contexts are uniquely determined, as shall see shortly.

The semantic combinators $^{>}$ and $^{<}$ in (10) use their continuation argument trivially; therefore, they also work for CPS^2 and for all other levels of the hierarchy. We need to give them more general schematic types, extending Figure 8 so it works at any level of the hierarchy:

$$(20)\qquad \frac{m_1 : (\sigma_2^{n_1} \to \{\alpha\}) \to \{\beta\} \quad m_2 : ((\sigma_2\sigma_1)^{n_2} \to \{\gamma\}) \to \{\alpha\}}{m_1{}^{<}m_2 : (\sigma_1^{\max(n_1,n_2)} \to \{\gamma\}) \to \{\beta\}}$$

$$(21)\qquad \frac{m_1 : ((\sigma_2\sigma_1)^{n_1} \to \{\alpha\}) \to \{\beta\} \quad m_2 : (\sigma_2^{n_2} \to \{\gamma\}) \to \{\alpha\}}{m_1{}^{>}m_2 : (\sigma_1^{\max(n_1,n_2)} \to \{\gamma\}) \to \{\beta\}}$$

We only need to change $(\!|\cdot|\!)$ to account for the two continuation arguments, and hence, two initial continuations:

$$(22)\qquad (\!|m|\!) \stackrel{\text{def}}{=} m(\lambda v.\, \lambda k_2.\, k_2\ v)(\lambda v.\, v)$$

The initial CPS^1 continuation $(\lambda v.\, \lambda k_2.\, k_2\ v)$ plugs its argument into the outer context; the initial outer context is the empty context. Schematically, $(\!|m|\!)$ may be diagrammed as $[_{k_2}[_{k_1} m]]$.

The two sets of quantifiers, level-1 and level-2, treat the inner and outer contexts differently. The remainder of this subsection presents several examples of computing denotations of sample sentences by using the lexical entries and the composition rules of Figure 10 and performing simplifications by β-reductions. As we shall see, the sequence of reductions for, say, $\mathsf{Someone}_1\ \mathsf{VP}$ can be diagrammed at a high level as follows:

$$(23)\qquad \begin{aligned} &[\![\mathsf{Someone}_1\ \mathsf{VP}.]\!] \\ &\quad = (\!|[\![\mathsf{Someone}_1\ \mathsf{VP}]\!]|\!) \\ &\quad = [_{k_2}[_{k_1}[\![\mathsf{Someone}_1\ \mathsf{VP}]\!]]] \\ &\quad \rightsquigarrow [_{k_2}\exists_n[_{k_1} n^{<}\ [\![\mathsf{VP}]\!]]] \end{aligned}$$

We hence see that it is the level-1 quantifiers that wedge themselves between the inner context k_1 and the outer context k_2. We also see that, if the VP contains only level-1 QNPs, they would quantify over $[_{k_1} n^{<} \ldots]$ giving the linear-scope reading. On the other hand, if the VP has a level-2 QNP, it will quantify over the outer context $[_{k_2}\exists_n[_{k_1} n^{<} \ldots]]$ yielding the inverse-scope reading. After this preview, we describe the computation of denotations in detail.

It is a simple exercise to show that $[_{\mathsf{M}}\ \mathsf{Someone}_1\ [_{\mathsf{VP}}\ \mathsf{saw\ everyone}_1\ ].]$ has the same linear-scope reading $\exists_0\forall_1\mathbf{see}\,\centerdot\,1\,\centerdot\,0$ as computed with the ordinary CPS, §2.2 – with essentially the same β-reductions shown in that section. It is also easy to see that $[_{\mathsf{M}}\ \mathsf{Someone}_2\ [_{\mathsf{VP}}\ \mathsf{saw\ everyone}_2\ ].]$ also has exactly the same denotation. The interesting cases are the sentences with different levels of

quantifiers. For example,

(24)

$[\![[_{\mathsf{M}}\ [_{\mathsf{S}}\ [_{\mathsf{NP}}\ \mathsf{Someone}_2]\ [_{\mathsf{VP}}\ [_{\mathsf{Vt}}\ \mathsf{saw}]\ [_{\mathsf{NP}}\ \mathsf{everyone}_1]]]\,.]]\!]$

$= (\lambda k_0.\,(\lambda k_1.\,\lambda k_2.\,\exists_0(k_1\ 0\ k_2))(\lambda x.$
$\quad(\lambda k_3.\,(\lambda k.\,k\ \mathbf{see})(\lambda f'.\,(\lambda k_1.\,\lambda k_2.\,k_1\ 1\ (\lambda v.\,k_2(\forall_1 v)))(\lambda x'.\,k_3\ (f'\,\centerdot\,x'))))$
$\quad(\lambda f.\,k_0\ (f\,\centerdot\,x))))$
$\quad(\lambda v.\,\lambda k_2.\,k_2\ v)(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_1.\,\lambda k_2.\,\exists_0(k_1\ 0\ k_2))(\lambda x.$
$\quad(\lambda k_2.\,(\lambda v.\,k_2(\forall_1 v))(\mathbf{see}\,\centerdot\,1\,\centerdot\,x)))$
$\quad(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_1.\,\lambda k_2.\,\exists_0(k_1\ 0\ k_2))(\lambda x.$
$\quad(\lambda k_2.\,k_2(\forall_1(\mathbf{see}\,\centerdot\,1\,\centerdot\,x))))$
$\quad(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_2.\,\exists_0(\forall_1(\mathbf{see}\,\centerdot\,1\,\centerdot\,0)))(\lambda v.\,v)$

$\rightsquigarrow \exists_0(\forall_1(\mathbf{see}\,\centerdot\,1\,\centerdot\,0))$

The result still shows the linear-scope reading, because $\mathsf{someone}_2$ quantifies over the wide context and so wins over the narrow-context quantifier $\mathsf{everyone}_1$. One may wonder how we chose the names of the quantified variables: 0 for $\mathsf{someone}_2$ and 1 for $\mathsf{everyone}_1$. The choice is clear from the final denotation: since it should have the type t^0 (that is, be closed), the schema for the corresponding $\mathsf{someone}_2$ must have been instantiated with $n = 0$. Therefore, $\forall_1(\mathbf{see}\,\centerdot\,1\,\centerdot\,0)$ must have the type t^1, which determines the schema instantiation for $\mathsf{everyone}_1$. One may say that 'names follow scope'. The variable names can also be chosen before β-reducing, while building the typing derivation, as demonstrated in §3.

We now make a different choice of lexical entries for the same quantifier words in the running example:

(25) $[\![[_{\mathsf{M}}\ [_{\mathsf{S}}\ [_{\mathsf{NP}}\ \mathsf{Someone}_1]\ [_{\mathsf{VP}}\ [_{\mathsf{Vt}}\ \mathsf{saw}]\ [_{\mathsf{NP}}\ \mathsf{everyone}_2]]]\,.]]\!]$

$= (\lambda k_0.\,(\lambda k_1.\,\lambda k_2.\,k_1\ 1\ (\lambda v.\,k_2(\exists_1 v)))(\lambda x.$
$\quad(\lambda k_3.\,(\lambda k.\,k\ \mathbf{see})(\lambda f'.\,(\lambda k_1.\,\lambda k_2.\,\forall_0(k_1\ 0\ k_2))(\lambda x'.\,k_3\ (f'\,\centerdot\,x'))))$
$\quad(\lambda f.\,k_0\ (f\,\centerdot\,x))))$
$\quad(\lambda v.\,\lambda k_2.\,k_2\ v)(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_1.\,\lambda k_2.\,k_1\ 1\ (\lambda v.\,k_2(\exists_1 v)))(\lambda x.$
$\quad(\lambda k_2.\,\forall_0(k_2\ (\mathbf{see}\,\centerdot\,0\,\centerdot\,x))))$
$\quad(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_2.\,(\lambda k_2.\,\forall_0(k_2\ (\mathbf{see}\,\centerdot\,0\,\centerdot\,1)))(\lambda v.\,k_2(\exists_1 v)))$
$\quad(\lambda v.\,v)$

$\rightsquigarrow (\lambda k_2.\,\forall_0((\lambda v.\,k_2(\exists_1 v))(\mathbf{see}\,\centerdot\,0\,\centerdot\,1)))$
$\quad(\lambda v.\,v)$

$\rightsquigarrow \forall_0(\exists_1(\mathbf{see}\,\centerdot\,0\,\centerdot\,1))$

We obtain the inverse-scope reading: $\mathsf{everyone}_2$ quantified over the higher, or wider, context and hence outscoped $\mathsf{someone}_1$. This outscoping is noticeable

already in the result of the first set of β-reductions, which may be diagrammed as $\forall_0[_{k_2}\exists_1[_{k_1}\mathbf{see}\,.\,0\,.\,[1]]]$. Since the universal quantifier eventually got the widest scope, the schema for $\mathsf{everyone}_2$ must have been instantiated with $n = 0$. Again, the choice of quantifier variable names is determined by quantifiers' scope.

Thus the continuation hierarchy lets us derive both linear- and inverse-scope readings of ambiguous sentences. The source of the quantifier ambiguity is squarely in the lexical entries for the quantifier words rather than in the rules of syntactic formation or semantic composition.

4.2 Continuation hierarchy in direct style

Like the ordinary CPS, the CPS hierarchy can also be built on demand. Therefore, we do not have to decide in advance the highest CPS level for our denotations, and be forced to rebuild our fragment's denotations should a new example call for yet a higher level. Rather, we build sentence denotations by combining parts with different CPS levels, or even not in CPS. The primitive parts, lexical entry denotations, may remain not in CPS (which is the case for all quantifier-free entries) or at the minimum needed CPS level, regardless of the level of other entries. The incremental construction of hierarchical CPS denotations – building up levels only as required – makes our fragment modular and easy to extend. It also relieves us from the tedium of dealing with unnecessarily high-level CPS terms.

Luckily, the semantic combinators $^<$ and $^>$ capable of combining the denotations of different CPS levels have already been defined. They are (14) and (15) in §2.3. The luck comes from the fact that the composition of CPS^1 denotations uses its continuation argument trivially, and therefore, works at any level of the CPS hierarchy. We only need to extend the schema for $(\!|\cdot|\!)$, in a regular way:

$$(\!|m|\!) \stackrel{\text{def}}{=} \begin{cases} m & \text{if } m : \{0\} \\ m(\lambda v.\, v) & \text{if } m : \{nn0\} \\ m(\lambda v.\, \lambda k.\, kv)(\lambda v.\, v) & \text{if } m : \{nnl_1l_1l_2l_20\} \\ \dots \end{cases} \tag{26}$$

Applying the schematic definition (26) requires a bit of explanation. If the term m has the type with no arrows, we should compute $(\!|m|\!)$ according to the first case, which requires m be of the type t^0. If m has the type that matches $\{nn0\}$, that is, $(t^n \to t^n) \to t^0$ for some n, we should use the second case, and so on. A term like $\lambda k.\, k$ (**leave . john**) of the schematic type $\{0\alpha\alpha\}$ may seem confusing: its type matches $\{nn0\}$ (with α instantiated to $\{0\}$ and n to 0) as well as the type $\{nnl_1l_1l_2l_20\}$ (with $\alpha = \{000\}$ and $n = l_1 = l_2 = 0$) and all further CPS types. We can compute $(\!|\lambda k.\, k$ (**leave . john**)$|\!)$ according to the second or any following case. The ambiguity is spurious however: whichever of the applicable equations we use, the result is the same – which follows from the fact that a CPS^i term which uses its continuation argument trivially is a $\mathrm{CPS}^{i'}$ term for

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
$\mathsf{M} \to \mathsf{S}\ .$	t^0	$(\!|[\![\mathsf{S}]\!]|\!)$
$\mathsf{S} \to \mathsf{NP\ VP}$	t^n or $(t^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{NP}]\!]^{<}[\![\mathsf{VP}]\!]$
$\mathsf{VP} \to \mathsf{Vt\ NP}$	et^n or $((et)^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{Vt}]\!]^{>}[\![\mathsf{NP}]\!]$
$\mathsf{NP} \to \mathsf{John}$	e	**john**
$\mathsf{NP} \to \mathsf{Mary}$	e	**mary**
$\mathsf{VP} \to \mathsf{left}$	et	**leave**
$\mathsf{Vt} \to \mathsf{saw}$	$e(et)$	**see**
$\mathsf{NP} \to \mathsf{everyone}_1$	$(e^{n+1} \to \{(n+1)\alpha\}) \to \{n\alpha\}$	$\lambda k_1.\, \lambda k_2.\, k_1\ n\ (\lambda v.\, k_2(\forall_n v))$
$\mathsf{NP} \to \mathsf{someone}_1$	$(e^{n+1} \to \{(n+1)\alpha\}) \to \{n\alpha\}$	$\lambda k_1.\, \lambda k_2.\, k_1\ n\ (\lambda v.\, k_2(\exists_n v))$
$\mathsf{NP} \to \mathsf{everyone}_2$	$(e^{n+1} \to \{\beta(n+1)\gamma\}) \to \{\beta n\gamma\}$	$\uparrow [\![\mathsf{everyone}_1]\!]$
$\mathsf{NP} \to \mathsf{someone}_2$	$(e^{n+1} \to \{\beta(n+1)\gamma\}) \to \{\beta n\gamma\}$	$\uparrow [\![\mathsf{someone}_1]\!]$

Fig. 11. Syntax and the multi-level direct-style continuation semantics for the small fragment: the merger of Figures 3 and 10. Lexical entries other than the quantifiers keep the simple denotations from Figure 3. Here α, β and γ are sequences of schematic meta-variables whose length is determined by the CPS level; β is two longer than γ.

all $i' \geq i$ [12]. As a practical matter, choosing the lowest-level instance of the schema (26) produces the cleanest derivation.

Figure 11 shows our new fragment.

The quantifier-free lexical entries have the simplest denotations and can be combined with CPS^n terms, $n \geq 0$. The quantifiers $\mathsf{everyone}_1$ and $\mathsf{someone}_1$ have the schematic denotations that can be used at the CPS^n level $n \geq 1$. The higher-level quantifiers are systematically produced by applying the $\uparrow$ combinator of the type $((e^{n+1} \to \{\alpha\}) \to \{\beta\}) \to ((e^{n+1} \to \{\gamma\alpha\}) \to \{\gamma\beta\})$ (where α and β have the same length and γ is one longer).

$$\uparrow m \stackrel{\text{def}}{=} \lambda k.\, \lambda k'.\, m(\lambda v.\, k\ v\ k') \tag{27}$$

With the entries in Figure (11), all sample derivations from §4 can be repeated in direct style with hardly any changes.

Our direct-style multi-level continuation semantics is essentially the same as that presented in [11]. We do not account for directionality in semantic types (since we use CFG or potentially CCG, rather than type-logical grammars) but we do account for the levels of quantified variables in types (whereas in [11], quantification was handled informally).

We have thus shown that the CPS hierarchy just as the ordinary CPS can be built on demand, without committing ourselves to any particular hierarchy level but raising the level if needed as a denotation is being composed. The result is the modular semantics, and much simpler and more lucid semantic derivations. From now on, we will use this multi-level direct style.

5 Scope islands and quantifier strength

We have used the continuation hierarchy to explain quantifier ambiguity between linear- and inverse-scope readings. We contend that the ambiguity arises because quantifier words are polysemous: they have multiple denotations corresponding to different levels of the CPS hierarchy. The higher the CPS level, the wider the quantifier scope.

We turn to two further problems. First, just quantifiers' competing with each other on their strength (CPS level) does not explain all empirical data. Some syntactic constructions such as embedded clauses come into play and restrict the scope of embedded quantifiers. That restriction however does not seem to spread to indefinites: "the varying scope of indefinites is neither an illusion nor a semantic epiphenomenon: it needs to be 'assigned' in some way" [2]. We shall use the CPS hierarchy to account for scope islands and to assign the varying scope to indefinites.

5.1 Scope islands

Like our running example "Someone saw everyone", two characteristic examples (4) and (5), repeated below, also have two quantifier words.

(28) That every boy left upset a teacher.

(29) Someone reported that John saw everyone.

These examples are not ambiguous however: (28) (the same as (4)) has only the inverse-scope reading, whereas (29) (the same as (5)) has only the linear-scope reading. The common explanation (see survey [2]) is that embedded tensed clauses are *scope islands*, preventing embedded quantifiers from taking scope wider than the island.

To analyze these examples, we at least have to extend our fragment with more lexical entries and with syntactic forms for clausal NPs, with the corresponding semantic combinators.[7] Figure 12 shows the additions. Most of them are straightforward. In particular, we generalize quantifying NPs like everyone to quantifying determiners like every. The determiner receives an extra (et) argument for its restrictor property, of the type of the denotation of a common noun.[8] Unlike Barker [5], we do not use choice functions in the denotations for the quantifier determiners. Instead, the denotation of the NP is obtained from the denotations of the Det and N by ordinary function application.

Just as quantifying NPs are polysemous, so are quantifying Dets on our analysis: there are weak (or level-1) forms every_1 and a_1 and strong (or level-2)

[7] If the domain of the semantic type t only contains the two truth values, we clearly cannot give an adequate denotation to embedded clauses: the domain is too small. Therefore, we now take the domain of t to be a suitable complete Boolean algebra.

[8] This is a simplification: generally speaking, the argument of a Det is not a bare common noun but a noun modified by PP and other adjuncts. Until we add PP to our fragment in §5.3, the simplification is adequate.

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
VP $\to$ Vs that S	et^n or $((et)^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{Vs}]\!]\ ^{>}(\!|[\![\mathsf{S}]\!]|\!)$
NP $\to$ that S	e	$\mathbf{That}\,.\,(\!|[\![\mathsf{S}]\!]|\!)$
NP $\to$ Det N	e^n or $(e^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{Det}]\!]\ [\![\mathsf{N}]\!]$
N $\to$ teacher	et	**teacher**
N $\to$ boy	et	**boy**
VP $\to$ disappeared	et	**disappear**
Vt $\to$ upset	$e(et)$	**upset**
Vs $\to$ report	$t(et)$	**report**
Det $\to$ every_1	$(et) \to (e^{n+1} \to \{(n+1)\alpha\}) \to \{n\alpha\}$	$\lambda z.\,\lambda k_1.\,\lambda k_2.\,k_1\ n\ (\lambda v.\ k_2\ (\forall_n(z\,.\,n \Rightarrow v)))$
Det $\to$ some_1, a_1	$(et) \to (e^{n+1} \to \{(n+1)\alpha\}) \to \{n\alpha\}$	$\lambda z.\,\lambda k_1.\,\lambda k_2.\,k_1\ n\ (\lambda v.\ k_2\ (\exists_n(z\,.\,n \wedge v)))$
Det $\to$ every_2	$(et) \to (e^{n+1} \to \{\beta(n+1)\gamma\}) \to \{\beta n\gamma\}$	$\lambda z.\uparrow([\![\mathsf{every}_1]\!]\ z)$
Det $\to$ some_2, a_2	$(et) \to (e^{n+1} \to \{\beta(n+1)\gamma\}) \to \{\beta n\gamma\}$	$\lambda z.\uparrow([\![\mathsf{some}_1]\!]\ z)$

Fig. 12. Syntax and the multi-level direct-style continuation semantics for the additional fragment.

forms every_2 and a_2. Stronger quantifiers outscope weaker ones. For example, $[_{\mathsf{M}}\ [_{\mathsf{S}}\ [\mathsf{a}_1\ \mathsf{boy}]\ [\mathsf{upset}\ [\mathsf{every}_2\ \mathsf{teacher}]]].]$ determines the inverse-scope reading $\forall_0(\mathbf{teacher}\,.\,0 \Rightarrow \exists_1(\mathbf{boy}\,.\,1 \wedge \mathbf{upset}\,.\,0\,.\,1))$.

Recall from Figure 11 how the matrix denotation $\mathsf{M} \to \mathsf{S}.$ is obtained from the denotation of the main clause: $[\![\mathsf{M}]\!] = (\!|[\![\mathsf{S}]\!]|\!)$. We see exactly the same pattern for the clausal NPs in the semantic operations corresponding to Vs that S and that S: in all the cases, the denotation of a clause is enclosed within $(\!|\cdot|\!)$, which is the semantic counterpart of the syntactic clause boundary. The typing rules for $(\!|\cdot|\!)$ in Figure 8 specify its result have the type t^0, as befits the denotation of a clause. The type t^0 is not a CPS type and hence $(\!|[\![\mathsf{S}]\!]|\!)$ cannot get hold of its context to quantify over. Therefore, if S had any embedded quantifiers, they can quantify only as far as the clause. The operation $(\!|\cdot|\!)$ thus acts as the scope delimiter, delimiting the context over which quantification is possible. (Incidentally, the same typing rules of $(\!|\cdot|\!)$ severely restrict how this scope-delimiting operation may be used within lexical entries. For example, $(\!|[\![\mathsf{VP}]\!]|\!)$ is ill-typed since VP does not have the type t^n or $(t^n \to \{\alpha\}) \to \{\beta\}$.)

In case of (28), we obtain the same denotation (30) no matter which lexical entry we choose for the embedded determiner, every_1 (31) or every_2 (32). The quantifier remains trapped in the clause and the sentence is not ambiguous. Incidentally, since all quantifier variables used within a clause will be quantified within the clause, their names can be chosen regardless of the names of other variables within the sentence. That's why the name 0 is reused in (30). Again, names follow scope. A similar analysis applies to (29).

(30) $\exists_0(\mathbf{teacher}\,.\,0 \wedge \mathbf{upset}\,.\,0\,.\,(\mathbf{That}\,.\,\forall_0(\mathbf{boy}\,.\,0 \Rightarrow \mathbf{leave}\,.\,0)))$

(31) $[\![[_{\mathsf{M}}\ [_{\mathsf{NP}}\ \mathsf{That}\ [_{\mathsf{S}}\ \mathsf{every}_1\ \mathsf{boy}\ \mathsf{left}]]\ [_{\mathsf{VP}}\ \mathsf{upset}\ [_{\mathsf{NP}}\ \mathsf{a}_1\ \mathsf{teacher}]].]]\!]$

(32) $[\![[_{\mathsf{M}}\ [_{\mathsf{NP}}\ \mathsf{That}\ [_{\mathsf{S}}\ \mathsf{every}_2\ \mathsf{boy}\ \mathsf{left}]]\ [_{\mathsf{VP}}\ \mathsf{upset}\ [_{\mathsf{NP}}\ \mathsf{a}_1\ \mathsf{teacher}]].]]\!]$

We have demonstrated that a scope island is an effect of the operation $(\!|\cdot|\!)$, which is the semantic counterpart of the syntactic clause boundary. In our analysis, each surface syntactic constituent still corresponds to a well-formed denotation, and each surface syntactic formation rule still corresponds to a semantic combinator. Our approach hence is directly compositional.

5.2 Wide-scope indefinites

Given that enclosing all clause denotations in $(\!|\cdot|\!)$ traps all quantifiers inside, how do indefinites manage to get out? And they do get out: "Indefinites acquire their existential scope in a manner that does not involve movement and is essentially syntactically unconstrained" [2, §3.2.1]. For example:

(33) Everyone reported that [Max and some lady] disappeared.

(34) Most guests will be offended [if we don't invite some philosopher].

(35) All students believe anything [that many teachers say].

Szabolcsi argued [2] that all these examples are ambiguous. In particular, in (33) (the same as (7)), either different people meant a different lady disappearing along with Max, or there is one lady that everyone reported as disappearing along with Max. Interestingly, the example

(36) Someone reported that [Max and every lady] disappeared.

is not ambiguous: there is a single reporter of the disappearance for Max and all ladies. The unambiguity of (36) is explained by the embedded clause's being a scope island, which prevents the universal from taking wide scope. The ambiguity of (33) leads us to conclude that indefinites, in contrast to universals, can scope out of clauses, complements and coordination structures. Szabolcsi [2] gives a large amount of evidence for this conclusion. Accordingly, our theory must first explain how anything can get out of a scope island, then postulate that only indefinites have this escaping ability.

The operation $(\!|\cdot|\!)$ that effects the scope island has the schematic type that can be informally depicted as $\mathrm{CPS}^i[t] \to \mathrm{CPS}^0[t]$ where

$$\mathrm{CPS}^i[t] = \{\alpha\} \quad \text{where the length of } \alpha \text{ is } 2^{i+1} - 1$$

Since the result of $(\!|m|\!)$ has a CPS^0 type, that is t, the result cannot get hold of any context. Hence we need a less absolutist version of $(\!|\cdot|\!)$ which merely lowers rather than collapses the hierarchy. We call that operation $(\!|\cdot|\!)_2$, of the informal schematic type $\mathrm{CPS}^{\leq 2}[t] \to \mathrm{CPS}^0[t]$ and $\mathrm{CPS}^{i+2}[t] \to \mathrm{CPS}^i[t]$ where $i \geq 1$. Whereas $(\!|m|\!)$ delimits all the contexts of m, $(\!|m|\!)_2$ delimits only the first two contexts of the hierarchy. Quantifiers within m of level 3 and higher will be able

to get hold of the context of $(\!|m|\!)_2$. One may think of $(\!|\cdot|\!)_2$ as the inverse of $\uparrow\uparrow$. The following example illustrates the lowering:

(37a) $[\![\mathsf{someone}_1\ \mathsf{left}]\!] = \lambda k_1.\,\lambda k_2.\,k_1\ (\mathbf{leave}\,.\,n)\ (\lambda v.\,k_2 \exists_n v)$

(37b) $(\!|[\![\mathsf{someone}_1\ \mathsf{left}]\!]|\!)_2 = \exists_0(\mathbf{leave}\,.\,0)$

(37c) $(\!|\uparrow [\![\mathsf{someone}_1\ \mathsf{left}]\!]|\!)_2 = \exists_0(\mathbf{leave}\,.\,0)$

(37d) $(\!|\uparrow\uparrow [\![\mathsf{someone}_1\ \mathsf{left}]\!]|\!)_2 = \lambda k_1.\,\lambda k_2.\,k_1\ (\mathbf{leave}\,.\,n)\ (\lambda v.\,k_2 \exists_n v)$

In (37a) and the identical (37d), the existential quantifies over the potentially wide context k_1. In (37b) and (37c), whose denotations are again identical, $\exists_0$ scopes just over $\mathbf{leave}\,.\,0$ and extends no further.

Why did we choose 2 as the number of contexts to delimit at the embedded clause boundary? Any number $i \geq 2$ will work, to explain the quantifier ambiguity within the embedded clause and wide-scope indefinites. We chose $i = 2$ for now pending analysis of more empirical data.

If (37) is the specification for $(\!|\cdot|\!)_2$, then (38) below is the implementation. It is derived from the schema (26) by cutting it off after the third line and inserting the generic lowering-by-two operation as the final default case.

(38) $$(\!|m|\!)_2 \stackrel{\text{def}}{=} \begin{cases} m & \text{if } m : \{0\} \\ m(\lambda v.\,v) & \text{if } m : \{nn0\} \\ m(\lambda v.\,\lambda k.\,kv)(\lambda v.\,v) & \text{if } m : \{nnl_1l_1l_2l_20\} \\ m(\lambda v.\,\lambda k.\,kv)(\lambda v.\,\lambda k.\,kv) & \text{otherwise} \end{cases}$$

It is easy to show that the definition (38) indeed satisfies (37). A useful lemma is the identity $(\uparrow m)(\lambda v.\,\lambda k.\,kv) = m$, easily verified from the definition (27) of $\uparrow$.

To make use of this lowering operation $(\!|\cdot|\!)_2$, we adjust the lexical entries in Figure 12 as shown in Figure 13. The main change is replacing $(\!|\cdot|\!)$ in the semantic composition rules for embedded clauses with $(\!|\cdot|\!)_2$. In other words, we now distinguish the main clause boundary from embedded clause boundaries. Figure 13 also reflects our postulate: only indefinites may be at the CPS level 3 and higher – not universals.

The typical example (33) can now be analyzed as follows (see Fig. 12 for the denotations of $\mathsf{disappeared}$ and report):

(39) $[_{\mathsf{M}}\ \mathsf{Everyone}_1\ \mathsf{reported\ that}\ [_{\mathsf{S}}\ \mathsf{Max\ and\ some}_i\ \mathsf{lady\ disappeared}].]$

When the level i of some_i is 1 or 2, the indefinite is trapped in the scope island.

(40a) $\forall_0 \mathbf{report}\,.\,(\exists_0(\mathbf{lady}\,.\,0 \wedge \mathbf{disappear}\,.\,(\mathbf{alongWith}\,.\,\mathbf{max}\,.\,0)))\,.\,0$

At the level $i = 3$, the indefinite scopes out of the clause but is defeated by the universal in the subject position, giving us another linear-scope reading, along the lines expounded in §2.2.

(40b) $\forall_0 \exists_1 \mathbf{lady}\,.\,1 \wedge \mathbf{report}\,.\,(\mathbf{disappear}\,.\,(\mathbf{alongWith}\,.\,\mathbf{max}\,.\,1))\,.\,0$

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
$\mathsf{VP} \to \mathsf{Vs\ that\ S}$	et^n or $((et)^n \to \{\alpha\}) \to \{\beta\}$	$[\![\mathsf{Vs}]\!] \mathbin{>} (\!|[\![\mathsf{S}]\!]|\!)_2$
$\mathsf{NP} \to \mathsf{that\ S}$	e^n or $(e^n \to \{\alpha\}) \to \{\beta\}$	$\mathbf{That} \mathbin{.} (\!|[\![\mathsf{S}]\!]|\!)_2$
$\mathsf{NP} \to \mathsf{NP}_1\ \mathsf{and}\ \mathsf{NP}_2$	e^n or $(e^n \to \{\alpha\}) \to \{\beta\}$	$\mathbf{alongWith} \mathbin{.} [\![\mathsf{NP}_1]\!] \mathbin{.} [\![\mathsf{NP}_2]\!]$
$\mathsf{N} \to \mathsf{max}$	e	$\mathbf{max}$
$\mathsf{N} \to \mathsf{lady}$	et	$\mathbf{lady}$
$\mathsf{Det} \to \mathsf{some}_3,\ \mathsf{a}_3$	$(et) \to (e^{n+1} \to \{\beta_3(n+1)\gamma\}) \to \{\beta_3 n\gamma\}$	$\lambda z. \uparrow [\![\mathsf{some}_2]\!]\ z$
$\mathsf{Det} \to \mathsf{some}_4,\ \mathsf{a}_4$	$(et) \to (e^{n+1} \to \{\beta_4(n+1)\gamma\}) \to \{\beta_4 n\gamma\}$	$\lambda z. \uparrow [\![\mathsf{some}_3]\!]\ z$

Fig. 13. Adjustments to the syntax and the multi-level direct-style continuation semantics for the additional fragment, to account for wide-scope indefinites. If the size of the sequence γ is j, the size of β_3 is $3(j+2)$ and of β_4 is $7(j+2)$.

Finally, some_4, lowered from level 4 to level 2 as it crosses the embedded clause boundary, has sufficient strength left to scope over the entire sentence.

(40c) $\quad \exists_0 \mathbf{lady} \mathbin{.} 0 \wedge \forall_1 \mathbf{report} \mathbin{.} (\mathbf{disappear} \mathbin{.} (\mathbf{alongWith} \mathbin{.} \mathbf{max} \mathbin{.} 0)) \mathbin{.} 1$

5.3 Inverse linking

Our analysis of inverse linking turns out quite similar to the analysis of wide-scope indefinites. We take the argument NP of a PP to be a scope island, albeit it is evidently a weaker island than an embedded tensed clause. We realize the island by an operation similar to $(\!|\cdot|\!)_2$. Therefore, a strong enough quantifier embedded in NP can escape and take a wide scope. That escaping from the island corresponds to inverse linking.

To demonstrate our analysis, we extend our fragment with prepositional phrases; see Figure 14. We add a category of N′ of nouns adjoined with PP. We generalize Det to take as its argument N′ rather than bare common nouns. For simplicity, we use the same $(\!|\cdot|\!)_2$ operation for the PP island as we used for the embedded-clause island. Recall that $(\wedge)$ is a constant of the type $t(tt)$ and we write the $\mathcal{D}_\mathcal{Q}$ expression $(\wedge) \mathbin{.} d_1 \mathbin{.} d_2$ as $d_1 \wedge d_2$.

The type of the quantificational determiners shows that a determiner takes a restrictor and a continuation, which may contain n other free variables. The determiner adds a new one, which it then binds. Although the denotations of determiners in Figure 14 bind the variables they themselves introduced, that property is not assured by the type system. For example, nothing prevents us from writing 'bad' lexical entries like $\lambda z. \forall_n z$ or 1. Although the type system will ensure that the overall denotation is closed, what a binder ends up binding will be hard to predict. It is an interesting problem to define 'good' lexical entries (with respect to scope) and codify the notion in the type system. This is the subject of ongoing work [18].

Syntax	Semantic type	Denotation $[\![\cdot]\!]$
$\mathsf{N'} \to \mathsf{N}$	$e^n \to n\alpha$	$\lambda x.\, \lambda k.\, k\ ([\![\mathsf{N}]\!]\ x)$
$\mathsf{N'} \to \mathsf{N'\ PP}$	$e^n \to n\alpha$	$\lambda x.\, (\wedge)^{>}([\![\mathsf{N'}]\!]\ x)^{>}([\![\mathsf{PP}]\!]\ x)$
$\mathsf{PP} \to \mathsf{from\ NP}$	$e^n \to n\alpha$	$\lambda x.\, (\!\|\mathbf{from}^{>}\ [\![\mathsf{NP}]\!]^{>} x\|\!)_2$
$\mathsf{NP} \to \mathsf{Det\ N'}$	$(e^n \to \{\alpha\}) \to \beta$	$[\![\mathsf{Det}]\!]\ [\![\mathsf{N'}]\!]$
$\mathsf{Det} \to \mathsf{every}_1$	$(e^{n+1} \to \{(n+1)n\alpha\beta\gamma\})$ $\to (e^{n+1} \to \{(n+1)\alpha\beta\}) \to \{\gamma\}$	$\lambda z.\, \lambda k_1.\, z\ n\ (\lambda x.\, \lambda k_2.$ $k_1\ n\ (\lambda v.\, k_2\ (\forall_n(x \Rightarrow v))))$
$\mathsf{Det} \to \mathsf{some}_1, \mathsf{a}_1$	$(e^{n+1} \to \{(n+1)n\alpha\beta\gamma\})$ $\to (e^{n+1} \to \{(n+1)\alpha\beta\}) \to \{\gamma\}$	$\lambda z.\, \lambda k_1.\, z\ n\ (\lambda x.\, \lambda k_2.$ $k_1\ n\ (\lambda v.\, k_2\ (\exists_n(x \wedge v))))$
$\mathsf{Det} \to \mathsf{no}_1$	$(e^{n+1} \to \{(n+1)n\alpha\beta\gamma\})$ $\to (e^{n+1} \to \{(n+1)\alpha\beta\}) \to \{\gamma\}$	$\lambda z.\, \lambda k_1.\, z\ n\ (\lambda x.\, \lambda k_2.$ $k_1\ n\ (\lambda v.\, k_2\ (\neg\,.\,\exists_n(x \wedge v))))$

Fig. 14. Adjustments to the syntax and the multi-level direct-style continuation semantics for the additional fragment, to account for prepositional phrases. The higher-level quantificational determiners are produced with the $\uparrow$ operations; see Figure 13 for illustration. If the size of the sequence α is j, the size of β is also j and the size of γ is $2j + 1$.

We analyze inverse linking thusly.

(41a) [NP No [N′ [N′ man] [PP from a foreign country]]] was admitted.

(41b) $\neg\,.\,\exists_0\mathbf{man}\,.\,0 \wedge (\exists_1\mathbf{country}\,.\,1 \wedge \mathbf{from}\,.\,1\,.\,0) \wedge \mathbf{admitted}\,.\,0$

(41c) $\exists_0\mathbf{country}\,.\,0 \wedge \neg\,.\,\exists_1\mathbf{man}\,.\,1 \wedge \mathbf{from}\,.\,0\,.\,1 \wedge \mathbf{admitted}\,.\,1$

The PP in (41a) contains an ambiguous quantifier. If the quantifier is weak, it is trapped in the PP island and gives the salient reading (41b). If the quantifier is strong enough to escape, the inverse-linking reading (41c) emerges. We thus reproduce quantifier ambiguity for QNP within NP and explain inverse linking.

6 Conclusions

We have given the first rigorous account of linear- and inverse-scope readings, scope islands, wide-scope indefinites and inverse linking based on the D&F continuation hierarchy. Quantifier ambiguity arises because quantifier words are polysemous, with multiple denotations corresponding to different levels of the hierarchy. The higher the level, the wider the scope. Embedded clauses and PPs create scope islands by lowering the hierarchy and trapping low-level quantifiers. Higher-level quantifiers (which we postulate only indefinites possess) can escape the island and take wider scope. The continuation hierarchy lets us assign scope to indefinites and universals and explain their differing scope-taking abilities.

Our analysis is directly compositional: each surface syntactic constituent corresponds to a well-formed denotation, and each surface syntactic formation rule corresponds to a *unique* semantic combinator.

We have shown how to build the continuation hierarchy modularly and on-demand, without committing ourselves to any particular hierarchy level but

raising the level if needed as a denotation is being composed. In particular, quantifier-free lexical entries have unlifted types and simple denotations.

We look forward to extending our analysis to other aspects of scope – how quantifiers interact with coordination (as in (6)), pronouns and polarity items – and to distributivity in universal quantification. We would also like to investigate if hierarchy levels can be correlated with Minimalism features or feature domains. Finally, we plan to extend our analyses of single sentences to discourse.

Acknowledgements We are very grateful to Chris Tancredi for many helpful suggestions and a thought-provoking conversation. We thank anonymous reviewers for their comments.

References

[1] Montague, R.: The proper treatment of quantification in ordinary English. In Thomason, R.H., ed.: Formal Philosophy: Selected Papers of Richard Montague. Yale University Press, New Haven (1974) 247–270
[2] Szabolcsi, A.: The syntax of scope. In: Handbook of Contemporary Syntactic Theory. Blackwell (2000) 607–634
[3] Szabolcsi, A.: Quantification. Cambridge University Press, Cambridge (2009)
[4] Reinhart, T.: Syntactic domains for semantic rules. In Guenthner, F., Schmidt, S.J., eds.: Formal Semantics and Pragmatics for Natural Languages. Reidel, Dordrecht (1979) 107–130
[5] Barker, C.: Continuations and the nature of quantification. Natural Language Semantics **10** (2002) 211–242
[6] de Groote, P.: Type raising, continuations, and classical logic. In van Rooy, R., Stokhof, M., eds.: Proceedings of the 13th Amsterdam Colloquium, Institute for Logic, Language and Computation, Universiteit van Amsterdam (2001) 97–101
[7] Shan, C.c.: Linguistic side effects. In Barker, C., Jacobson, P., eds.: Direct Compositionality, New York, Oxford University Press (2007) 132–163
[8] Bekki, D., Asai, K.: Representing covert movements by delimited continuations. In: Proceedings of the 6th International Workshop on Logic and Engineering of Natural Language Semantics, Japanese Society of Artificial Intelligence (2009)
[9] Bernardi, R., Moortgat, M.: Continuation semantics for the Lambek-Grishin calculus. Information and Computation **208** (2010) 397–416
[10] Shan, C.c.: Inverse scope as metalinguistic quotation in operational semantics. In Yoshimoto, K., ed.: Proceedings of the 4th International Workshop on Logic and Engineering of Natural Language Semantics, Japanese Society of Artificial Intelligence (2007) 167–178
[11] Shan, C.c.: Delimited continuations in natural language: Quantification and polarity sensitivity. In Thielecke, H., ed.: CW'04: Proceedings of the 4th ACM SIGPLAN Continuations Workshop. Number CSR-04-1 in Tech. Rep., School of Computer Science, University of Birmingham (2004) 55–64

[12] Danvy, O., Filinski, A.: Abstracting control. In: Proceedings of the 1990 ACM Conference on Lisp and Functional Programming, New York, ACM Press (1990) 151–160
[13] Barker, C., Shan, C.c.: Donkey anaphora is in-scope binding. Semantics and Pragmatics **1** (2008) 1–46
[14] Champollion, L., Tauberer, J., Romero, M.: The Penn Lambda Calculator: Pedagogical software for natural language semantics. In King, T.H., Bender, E.M., eds.: Proceedings of the Workshop on Grammar Engineering Across Frameworks, Stanford, CA, Center for the Study of Language and Information (2007) 106–127
[15] Barker, C., Shan, C.c.: Types as graphs: Continuations in type logical grammar. Journal of Logic, Language and Information **15** (2006) 331–370
[16] Rompf, T., Maier, I., Odersky, M.: Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform. In Hutton, G., Tolmach, A.P., eds.: ICFP '09: Proceedings of the ACM International Conference on Functional Programming, New York, ACM Press (2009) 317–328
[17] Partee, B.H., Rooth, M.: Generalized conjunction and type ambiguity. In Bäuerle, R., Schwarze, C., von Stechow, A., eds.: Meaning, Use and Interpretation of Language. Walter de Gruyter, Berlin (1983) 361–383
[18] Kameyama, Y., Kiselyov, O., Shan, C.c.: Combinators for impure yet hygienic code generation. In Chin, W.N., Hage, J., eds.: PEPM, New York, ACM Press (2014) 3–14