Importing Montagovian Dynamics into Minimalism

Gregory M. Kobele

University of Chicago
kobele@uchicago.edu

Abstract. Minimalist analyses typically treat quantifier scope interactions as being due to movement, thereby bringing constraints thereupon into the purview of the grammar. Here we adapt De Groote's continuation-based presentation of dynamic semantics to minimalist grammars. This allows for a simple and simply typed compositional interpretation scheme for minimalism.

1 Introduction

Minimalist grammars [33] provide a mildly context sensitive perspective on mainstream chomskyian linguistic theory. Although semantic interpretation in chomskyian linguistics is traditionally viewed as operating on derived structures [17], this was faithfully reformulated in terms of a compositional semantics over derivation trees in [23]. There, in keeping with the treatment of pronouns as denoting variables, the standard semantic domains (of individuals E and of propositions T) were paramaterized with the set G of assignment functions, and a function $\leftthreetimes : \left[E^G \to T^G \to E^G \to T^G\right]$ which behaves in a manner similar to lambda abstraction was defined. Around the same time, a continuation-based reinterpretation of dynamic semantics using the simply typed lambda calculus was presented [8]. Instead of being variables, pronouns are treated there as (lifted) choice functions over *contexts*, which parameterize the type o of propositions.[1]

In this paper, we adopt the choice function treatment of pronouns [8], and reformulate the non-canonical semantics of minimalist grammars [23] in these terms. This allows for a simply typed and variable free (§3.3) presentation of minimalist semantics, within which constraints on quantifier scoping are most naturally formulated syntactically. This constrasts with a previous semantic interpretation scheme for (a logical reconstruction of) minimalist grammars [2, 27], which, using the lambda-mu calculus [29] to represent the meanings of sentences, treated scope taking as a consequence of different reduction orders. (And which, as a consequence, was not able to account for the various seemingly syntactic constraints on scope.)

The main 'data' to be covered include the following sentences.

[1] A greater similarity with [23] emerges if we do not lift the pronouns of [8], but instead treat them as denoting functions from contexts to individuals. Then both propositions *and* individuals must be parameterized by contexts.

1. Every boy believed that every man believed that he smiled.
2. Some man believed that every woman smiled.
3. Some man believed every woman to have smiled.

The interest in these sentences is as follows. Sentence 1 has a reading in which the pronoun *he* is bound by the matrix subject *every boy*. This can be 'straightforwardly' dealt with if *he* denotes a variable, as long as this variable has a different name than the one bound by the embedded subject. If we identify variable names with movement features (as we will here), this cannot be done. Sentences 2 and 3 differ in the scope taking behavior of the quantified noun phrase *every woman*. In 2, this QNP must scope under the matrix subject, while in 3 either scope order is possible.

The remainder of this paper is structured as follows. Section §2 introduces the minimalist grammar formalism. Section 3 presents a semantics for this formalism in terms of the simply typed lambda calculus. In §4, a grammar fragment is presented which allows for quantifiers to scope out of arbitrarily many non-finite clauses, but not past a tense clause boundary, as is the received wisdom in the linguistic literature [21]. Section 5 is the conclusion.

2 Minimalist Grammars

Minimalist grammars make use of two syntactic structure building operations; binary **merge** and unary **move**. **Merge** acts on its two arguments by combining them together into a single tree. The operation **move** rearranges the pieces of its single and syntactically complex argument. The generating functions **merge** and **move** are not defined on all objects in their domain. Whether a generating function is defined on a particular object in its domain (a pair of expressions in the case of **merge**, or a single expression in the case of **move**) is determined solely by the syntactic categories of these objects. In minimalist grammars, syntactic categories take the form of 'feature bundles', which are simply finite sequences of features. The currently accessible feature is the feature at the beginning (leftmost) position of the list, which allows for some features being available for checking only after others have been checked. In order for **merge** to apply to arguments Γ and Δ, the heads of both expressions must have matching first features in their respective feature bundles. These features are eliminated in the derived structure which results from their merger. In the case of **move**, the head of its argument Γ must have a feature matching a feature of the head of one of its subconstituents' Δ. In the result, both features are eliminated. Each feature type has an attractor and an attractee variant (i.e. each feature is either positive or negative), and for two features to match, one must be positive and the other negative. The kinds of features relevant for the **merge** and **move** operations are standardly taken for convenience to be different. For **merge**, the attractee feature is a simple categorial feature, written `x`. There are two kinds of attractor features, `=x` and `x=`, depending on whether the selected expression is to be merged on the right (`=x`) or on the left (`x=`). For the **move** operation, there is a single attractor feature, written `+y`, and two attractee features, `-y` and $\ominus$`y`, depending on whether the movement is overt (`-y`) or covert ($\ominus$`y`).

A lexical item is an atomic pairing of form and meaning, along with the syntactic information necessary to specify the distribution of these elements in more complex expressions. We write lexical items using the notation $\langle \sigma, \delta \rangle$, where σ is a lexeme, and δ is a feature bundle.

Complex expressions are written using the notation of [33] for the 'bare phrase structure' trees of [5]. These trees are essentially X-bar trees without phrase and category information represented at internal nodes. Instead, internal nodes are labeled with 'arrows' `>` and `<`, which point to the head of their phrase. A tree of the form $[_{<} \ \alpha \ \beta]$ indicates that the head is to be found in the subtree α, and we say that α projects over β, while one of the form $[_{>} \ \alpha \ \beta]$ that its head is in β, and we say that β projects over α. Leaves are labeled with lexeme/feature pairs (and so a lexical item $\langle \alpha, \delta \rangle$ is a special case of a tree with only a single node). The head of a tree t is the leaf one arrives at from the root by following the arrows at the internal nodes. If t is a bare phrase structure tree with head $\textsc{h}$, then I will write $t[\textsc{h}]$ to indicate this. (This means we can write lexical items $\langle \alpha, \delta \rangle$ as $\langle \alpha, \delta \rangle[\langle \alpha, \delta \rangle]$.) The **merge** operation is defined on a pair of trees t_1, t_2 if and only if the head of t_1 has a feature bundle which begins with either `=x` or `x=`, and the head of t_2 has a feature bundle beginning with the matching `x` feature. The bare phrase structure tree which results from the merger of t_1 and t_2 has t_1 projecting over t_2, which is attached either to the right of t_1 (if the first feature of the head was `=x`) or to the left of t_1 (if the first feature of the head was `x=`). In either case, both selection features are checked in the result.

$$\textbf{merge}(t_1[\langle \alpha, \texttt{=x}\delta \rangle], t_2[\langle \beta, \texttt{x}\gamma \rangle]) = \begin{array}{ccc} & \texttt{<} & \\ & \diagup \quad \diagdown & \\ t_1[\langle \alpha, \delta \rangle] & & t_2[\langle \beta, \gamma \rangle] \end{array}$$

$$\textbf{merge}(t_1[\langle \alpha, \texttt{x=}\delta \rangle], t_2[\langle \beta, \texttt{x}\gamma \rangle]) = \begin{array}{ccc} & \texttt{>} & \\ & \diagup \quad \diagdown & \\ t_2[\langle \beta, \gamma \rangle] & & t_1[\langle \alpha, \delta \rangle] \end{array}$$

If the selecting tree is both a lexical item and an affix (which I notate by means of a hyphen preceding/following the lexeme in the case of a suffix/prefix), then head movement is triggered from the head of the selected tree to the head of the selecting tree.

$$\textbf{merge}(\langle \text{-}\alpha, \texttt{=x}\delta \rangle, t_2[\langle \beta, \texttt{x}\gamma \rangle]) = \begin{array}{ccc} & \texttt{<} & \\ & \diagup \quad \diagdown & \\ \langle \beta\text{-}\alpha, \delta \rangle & & t_2[\langle \epsilon, \gamma \rangle] \end{array}$$

The operation **move** applies to a single tree $t[\langle \alpha, \texttt{+y}\delta \rangle]$ only if there is *exactly one* leaf ℓ in t with matching first feature `-y` or $\ominus$`y`.[2] This is a radical version of the shortest move constraint [5], and will be called the SMC – it requires that an

[2] Other constraints have been explored in [12].

expression move to the first possible landing site. If there is competition for that landing site, the derivation crashes (because the losing expression will have to make a longer movement than absolutely necessary). If it applies, **move** moves the maximal projection of ℓ to a newly created specifier position in t (overtly, in the case of $\texttt{-y}$, and covertly, in the case of $\ominus\texttt{y}$), and deletes both licensing features. To make this precise, let $t\{t_1 \mapsto t_2\}$ denote the result of replacing all subtrees t_1 in t with t_2, for any tree t, and let ℓ_t^M denote the maximal projection of ℓ in t, for any leaf ℓ.

$$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta\rangle]) = \begin{tikzcd}[column sep=tiny, row sep=small] & > \arrow[dl, dash] \arrow[dr, dash] & \\ t'[\langle\beta,\gamma\rangle] & & t[\langle\alpha,\delta\rangle]\{t' \mapsto \langle\epsilon,\epsilon\rangle\} \end{tikzcd}$$
$$(\text{where } t' = \langle\beta, \texttt{-y}\gamma\rangle_t^M)$$

$$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta\rangle]) = \begin{tikzcd}[column sep=tiny, row sep=small] & > \arrow[dl, dash] \arrow[dr, dash] & \\ \langle\epsilon,\gamma\rangle & & t[\langle\alpha,\delta\rangle]\{t' \mapsto t'[\langle\beta,\epsilon\rangle]\} \end{tikzcd}$$
$$(\text{where } t' = \langle\beta, \ominus\texttt{y}\gamma\rangle_t^M)$$

2.1 The Shortest Move Constraint

Minimalist grammars with the shortest move constraint were proven [28] to be weakly equivalent to multiple context free grammars [31]. The proof that minimalist languages are contained in the MCFLs proceeds by constructing an equivalent MCFG whose derivation trees are identical to those of the minimalist grammar (modulo a projection). Each component of a derived tuple of strings in the target MCFG corresponds to either a moving subexpression in the MG or to the fixed head and non-moving material around it. The shortest move constraint ensures that there is a finite upper bound on the number of possible moving subexpressions, and thus on the dimension of the target MCFG.

The derived trees of Minimalist Grammars, while not corresponding to so natural a class due to non-logical restrictions (such as the distribution of traces), are contained [26] in the tree languages derivable by multiple regular tree grammars, which are MCFGs where the derived tuples contain trees instead of strings [10].

These observations motivate the idea that, at least in the context of the SMC, the natural data structure for the objects derived by minimalist grammars are tuples, where all positions but the first are indexed by feature types. An alternative presentation of this data structure is as a *store*, in the sense of [6], which is a pair of an object and a finite map from feature types to objects. In the case of syntax, the objects are trees. In the case of semantics, they will turn out to be simply typed lambda terms, as explained next.

2.2 Derivations

A derivation tree is an element of the term language over the ranked alphabet $A_0 \cup A_1 \cup A_2$, where $A_0 = \mathit{Lex}$ is the set of nullary symbols, $A_1 = \{\mathbf{move}\}$

is the set of unary symbols, and $A_2 = \{\mathbf{merge}\}$ the set of binary symbols. As a consequence of the translation of minimalist grammars into multiple context free grammars [28, 16], and as described in [26], the set of derivation trees in a minimalist grammar of an expression with unchecked feature string γ at the root and no features anywhere else is regular.

For reasons of space (and because the derivation tree is more informative than any single derived tree), we will present only derivation trees for the expressions in this paper.

3 Minimalist Semantics

Here we present a rule-by-rule semantic interpretation scheme for minimalist grammars. The denotation of a syntactic object t is a pair of a simply typed lambda term and a quantifier store. A quantifier store is a partial function from feature types to simply typed terms. The idea is that a syntactic object t with a moving subexpression $t' = \langle\beta, \text{-y}\gamma\rangle_t^M$ has a quantifier store $\mathcal{Q}$ such that $\mathcal{Q}(y)$ is the stored meaning of t'.

We use lower case greek letters $(\alpha, \beta, \ldots)$ to stand for denotations of syntactic objects (pairs of simply typed lambda terms and quantifier stores), and the individual components of these denotations will be referred to with the corresponding roman letters in lowercase for the lambda term component, and in uppercase for the store component.[3] Thus, $\alpha = \langle a, A\rangle$.

We treat lexical items as being paired with the empty store.

Quantifier stores With $\emptyset$ we denote the empty quantifier store, such that for all feature types F, $\emptyset(f)$ is undefined. If two quantifier stores $\mathcal{Q}_1, \mathcal{Q}_2$ have disjoint domains,[4] then $\mathcal{Q}_1 \vee \mathcal{Q}_2$ denotes the store such that:

$$\mathcal{Q}_1 \vee \mathcal{Q}_2(f) := \begin{cases} \mathcal{Q}_1(f) & \text{if defined} \\ \mathcal{Q}_2(f) & \text{otherwise} \end{cases}$$

Let $\mathcal{Q}$ be a quantifier store, and F, G feature types. Then $\mathcal{Q}/f$ is the quantifier store just like $\mathcal{Q}$ except that it is undefined on F, $\mathcal{Q}[f := \alpha]$ is the store just like $\mathcal{Q}$ except that it maps F to α, and $\mathcal{Q}_{f\leftarrow g}$ is the store just like $\mathcal{Q}/g$ except that it maps F to whatever $\mathcal{Q}$ mapped G to.

Variable naming Our approach to variable naming is from [32], and is based on the observation that, in the context of the SMC, the type of the next feature of a moving expression uniquely identifies it. We will thus need no free individual variables (those of type e) other than these, and thus we subscript them with feature types.

[3] Lower case greek letters also have been used to stand for feature sequences, lexemes, etc. This overloading is hoped to be clear from context.

[4] If this is not the case, then the syntactic expressions they correspond to cannot be syntactically merged, as this would result in a violation of the SMC.

3.1 Merge

There are two possible semantic reflexes of syntactic merger. The first case is function application from one argument to the other (in a type driven manner). In the second case one of the arguments is a moving expression, and the other denotes a function from individuals. Here we insert it into the store indexed by the next licensee feature type it will move to check. The other argument is applied to a variable of the same name as the index under which the moving argument was stored. We denote these two semantic operations **mergeApp** and **mergeStore**.[5]

$$\mathbf{mergeApp}(\alpha,\beta) = \begin{cases} \langle a(b),\ A \cup B\rangle \\ \quad\quad\text{or} \\ \langle b(a),\ B \cup A\rangle \end{cases}$$

$$\mathbf{mergeStore}(\alpha,\beta) = \langle a(x_f),\ A \cup B[f := b]\rangle$$
$$\text{(where } \beta\text{'s next feature is } \texttt{-f}\text{)}$$

3.2 Move

There are multiple possible semantic reflexes of syntactic movement as well.[6] The first, **moveEmpty**, is used when the moving expression has previously taken scope. The second, **moveLater**, will be used when the moving expression is going to take scope in a later position. The third, **moveNow**, is used when the moving expression is going to take scope in this position. Here, the appropriate variable is abstracted over, and the resulting predicate is given as argument to the stored expression. In the below, we assume `-f` to be the feature checked by this instance of **move**.

$$\mathbf{moveEmpty}(\alpha) = \langle a, A\rangle$$
$$\text{where } A \text{ is undefined at } f$$

$$\mathbf{moveLater}(\alpha) = \langle (\lambda x_f.a)(x_g), A_{g\leftarrow f}\rangle$$
$$\text{for } \texttt{-g} \text{ the next feature to check}$$

$$\mathbf{moveNow}(\alpha) = \langle A(f)(\lambda x_f.a),\ A/f\rangle$$

[5] Note that the condition on **mergeStore** refers to the features of the moving expression. This information is present in the categories used in the MCFG translation of a minimalist grammar, and is a finite state bottom up relabeling of the standard minimalist derivation tree. Thus while not a homomorphic interpretation of minimalist derivations, it is a homomorphic interpretation of a finite state relabeling of minimalist derivations, or a transductive interpretation of minimalist derivations.

[6] [24] argues for the inclusion of function composition, in order to account for inverse linking constructions, which are beyond the scope of this paper.

3.3 Going Variable Free

As observed in [23], remnant movement wreaks havoc with the interpretation of movement dependencies presented here. The problem is that we can end up with a term in which variables remain free, with the lambda which was supposed to have bound them to their right. The reason this problem exists is that the semantics presented here treats each moving expression independently of all others, while the existence of remnant movement forces us to 'coordinate' the interpretations of two moving expressions where one contains the base position of the other. There are various strategies for resolving this difficulty [23], however, the one which suggests itself in the present simply typed context is nice because it simply eliminates free variables (named after features or not).

The basic idea is straightforward: an expression $\alpha = \langle a, A\rangle$ 'abbreviates' an expression $\alpha' = \langle \lambda x_{f_1}, \ldots, x_{f_n}.a, A\rangle$, where the domain of A is exactly the set of features $f_1, \ldots, f_n$. In other words, we take the 'main denotation' of an expression to have the variables expressions in the store may have introduced in it already and always bound. For any store A, define $\mathsf{var}(A)$ to be a fixed enumeration of the domain of A (viewed as variables), and $\Lambda(A).\ \phi$ to be a prefix of lambda abstractions over the domain of A viewed as variables. Then the **mergeApp** rule can be given as follows:

$$\mathbf{mergeApp}(\alpha, \beta) = \begin{cases} \Lambda(A \cup B).\ a(\mathsf{var}(A))(b(\mathsf{var}(B))),\ \ A \cup B\rangle \\ \qquad\quad \text{or} \\ \langle \Lambda(B \cup A).b(\mathsf{var}(B))(a(\mathsf{var}(A))),\ \ B \cup A\rangle \end{cases}$$

The rules **moveEmpty** and **moveLater** are also simply recast in these terms.

$$\mathbf{moveEmpty}(\alpha) = \langle \Lambda(A).a(\mathsf{var}(A)), A\rangle$$
$$\mathbf{moveLater}(\alpha) = \langle \Lambda(A_{j\leftarrow i}).(\lambda x_i.a(\mathsf{var}(A)))(x_j), A_{j\leftarrow i}\rangle$$

Not everything is so straightforward, unfortunately. What are we to do with the **mergeStore** rule, when the expression whose denotation is to be stored itself contains moving pieces? An example is verb phrase topicalization (in a sentence like *"Use the force, Luke will"*). In a typical minimalist analysis of this construction, the lexical item *will* merges with the verb phrase t_k *use the force*, which itself contains a moving expression, the subject *Luke*$_k$. Thus, under our 'variable free' view, we have a VP with main denotation of type et (because its store contains the denotation of *Luke*), but *will* is of type tt. There is a natural resolution to this (self-inflicted) problem, which implements the transformational observation that 'remnant movement obligatorilly reconstructs' [4]. We split the **mergeStore** rule into **mergeStoreMB**,[7] which simply stores expressions with empty stores, and **mergeStoreHO**,[8] which stores the denotation of a merging

[7] For '**M**onadic **B**ranching' [22].
[8] For '**H**igher **O**rder'.

expression with moving pieces, and inserts a higher order variable.

$$\begin{aligned}\textbf{mergeStoreMB}(\alpha, \langle b, \emptyset\rangle) &= \langle \Lambda(X).\ a(\mathsf{var}(A))(x_i),\ X\rangle \\ &\quad (\text{for } X = A[i := b], \text{ and } \mathtt{ty}(b) = (\mathtt{ty}(x_i) \to \gamma) \to \delta.) \\ \textbf{mergeStoreHO}(\alpha, \beta) &= \langle \Lambda(X).\ a(\mathsf{var}(A))(x_i(\mathsf{var}(B))),\ X\rangle \\ &\quad (\text{for } X = A \cup B[i := b], \text{ and } \mathtt{ty}(x_i) = \mathtt{ty}(b).)\end{aligned}$$

The quantifier store holds both generalized quantifiers, as before, as well as relations like $\lambda x.x$ *uses the force*. This latter type of expression seems to be used as an argument of some operator (such as a focus operator). Accordingly, we adjust the **moveNow** rule to allow a type driven retrieval scheme:

$$\begin{aligned}\textbf{moveNowFunc}(\alpha) &= \langle \Lambda(X).\ A(i)(\lambda x_i.a(\mathsf{var}(A)),\ X\rangle \qquad (\text{for } X = A/i) \\ \textbf{moveNowArg}(\alpha) &= \langle \Lambda(X).\ \lambda x_i.a(\mathsf{var}(A))(A(i)),\ X\rangle \qquad (\text{for } X = A/i)\end{aligned}$$

This is not a particularly interesting semantic treatment of remnant movement, as it simply implements obligatory 'semantic reconstruction' [7]. To properly implement linguistic analyses, it seems that some lexical items need access to the stores of their arguments. This would allow us to assign the following interpretation to a topicalization lexical item, which checks the topic feature of a moving expression, and returns a bipartite structure of the form $\mathtt{top}(\phi)(\psi)$, which asserts that ψ is the topic of ϕ: $[\![\langle\epsilon, \mathtt{=t\ +top\ c}\rangle]\!](\alpha) = \Lambda(A).\mathtt{top}(\lambda x_{top}.a(v(A)))(x_{top})$. This and alternatives need to be worked out further.

3.4 Determining Quantifier Scope

It is natural to view the minimalist grammar operations as pairs of syntactic and semantic functions. Thus, for example, we have both $\langle \mathtt{merge}, \textbf{mergeApp}\rangle$ and $\mathtt{merge}, \textbf{mergeStore}\rangle$. The semantic interpretation rules implement a reconstruction theory of quantifier scope [19], according to which the positions in which a quantified noun phrase can take scope are exactly those through which it has moved.

A shortcoming of this proposal as it now stands is that the yield language of the well formed derivation trees is ambiguous (in contrast to the uninterpreted minimalist grammar system [15]). Because minimalist grammar derivation tree languages are closed under intersection with regular sets [14, 25], any regular strategy for determining which of the possible positions a quantified noun phrase should be interpreted in can be used to give control over scope taking back to the lexicon. A simple such strategy is to assign to a licensee feature which may be used at **moveNow** a particular diacritic. A moving expression is then required to take scope at the highest (derivational) position permitted in which it checks its features, or in its merged position, should no other possibility obtain. We will adopt this proposal here, writing a licensee feature of type F which requires scope taking with a hat ($\mathtt{-\hat{f}}$ or $\ominus\mathtt{\hat{f}}$), and one which does not without ($\mathtt{-f}$ or $\ominus\mathtt{f}$). This move restores the functionality of the relation between sequences of lexical items and well-formed derivations, at the cost of increasing the size of the lexicon (by an additive factor).

3.5 Lexical Interpretations

A model is given by a set of atomic individuals E, a set of propositional interpretations T, and an interpretation function $\mathcal{I}$ assigning to each lexical item a model theoretic object in a set built over E and T. We assume throughout that $\mathcal{I}$ assigns to lexical items denotations of the 'standard' type; common nouns denote functions from individuals to propositions, n-ary verbs functions from n individuals to propositions, determiners relations between common noun denotations, etc.

We call the type of atomic individuals e, and that of propositions as o. Following [8], let the type of a (discourse) context be γ, and a sentence (to be revised shortly) to evaluate to a proposition only in a context (i.e. to be a function from contexts to propositions). Contexts will serve here to provide the input to pronoun resolution algorithms. For simplicity, we will treat them as lists of individuals (with [8], but see [3] for a more sophisticated treatment). The operation of updating a context $c \in \gamma$ with an individual a is written $a :: c$. Pronoun resolution algorithms *select* individuals from contexts, and are generically written $\texttt{sel}$.[9] We will assume that the individual selected from the context is actually present in the context ($\texttt{sel}(\gamma) \in \gamma$), leaving aside the question of empty contexts, and more precise conditions on the identity of the selected individual.

To deal with dynamic phenomena in his system, [8] lifts the type of a sentence once again to be a function from contexts (of type γ) and discourse continuations (functions of type γo) to propositions; an expression of type $\gamma(\gamma o)o$, which we will abbreviate as t. The idea is that a sentence is interpreted as a function from both its left context c and its right context ϕ to propositions.

With the exception of verb denotations, and expressions (such as relative pronouns) which are analysed there as taking verb denotations as arguments, we are able to simply take over the denotations assigned to lexical items from [8].

The style of analysis popular in the minimalist syntactic literature (and followed here), makes less straightforward a homomorphic relation between syntactic and semantic types. For example, the subject of a sentence is typically selected by a 'functional head', i.e. a lexical item other than the verb.

Relations We first look at the denotations of n-ary relation denoting lexical items, such as nouns and verbs. While nouns are treated here just as in [8], we treat verbs as on par with nouns, not, as does [8], as functions which take generalized quantifier denotations as arguments. This is because, in the minimalist grammar system, scope is dealt with by movement, not by modifying the verbal denotation (as is standard in categorial approaches to scope [18]).

[9] [9] gives pronoun resolution algorithms an additional property argument, and proposes that they select an individual with that property from the context.

A common noun, such as *monkey*, which is interpreted in the model as a function **monkey** of type *eo* mapping entities to true just in case they are monkeys, denotes a function of type *et*:

$$[\![monkey]\!](x)(c)(\phi) := \mathbf{monkey}(x) \wedge \phi(c)$$

Similarly, a transitive verb, such as *eat*, interpreted as a function $\mathbf{eat} : eeo$ mapping pairs of entities to true just in case the second ate the first, denotes a function of type *eet*:

$$[\![eat]\!](x)(y)(c)(\phi) := \mathbf{eat}(x)(y) \wedge \phi(c)$$

In general, given a function $f : \underbrace{e \cdots e}_{n \text{ times}} o$, we lift it to $\textsc{lift}(f) : \underbrace{e \cdots e}_{n \text{ times}} t$:

$$\textsc{lift}(f)(x_1) \cdots (x_n)(c)(\phi) := f(x_1) \cdots (x_n) \wedge \phi(c)$$

The conjunction and conjunct $\phi(c)$ common to all such predicates cashes out the empirical observation that propositions in a discourse combine conjunctively [30].

We adopt the following convention regarding arguments of type o (propositions): they are lifted to arguments of type t, and are passed as arguments the left and right context parameters of the lifted lexical item. As an example, take **believe** to be interpreted as a function of type *oeo*; a relation between a proposition (the belief) and an individual (the believer).

$$\textsc{lift}(\mathbf{believe})(S)(x)(c)(\phi) := \mathbf{believe}(S(c)(\phi))(x) \wedge \phi(c)$$

This illustrates a difficulty with the dynamic aspects of the system; it is not obvious how to allow a context to pick up referents contained in a different branch (here the propositional argument to **believe**) than which the continuation is (here in a position 'c-commanding' this argument). Should the propositional argument to **believe** contain a proper name, for example, this should be able to be found in the context of the remainder of the sentence. We do not dwell further on this difficulty here (though see footnote 11).

Noun phrases Traditional noun phrases (which are here, in line with [1], called 'determiner phrases', or DPs) such as *Mary*, *he*, or *every monkey*, denote functions $g : (et)t$. Proper names are interpreted as generalized quantifiers of type $(eo)o$, which are then lifted to the higher type via the operation GQ:

$$\textsc{gq}(G)(P)(c)(\phi) = G(\lambda x_e.\ P(x)(c)(\lambda d.\phi(x::d)))$$

Note that the individual 'referred to' by the generalized quantifier is incorporated into the context $(x::c)$ of the continuation of the sentence (ϕ). This permits future pronouns to pick up this individual as a possible referent. As an example, $\mathbf{Mary} = \lambda P_{eo}.P(m)$. And so $\textsc{gq}(\mathbf{Mary}) = \lambda c, \phi.P(m)(c)(\lambda d.\phi(m::d))$.

As mentioned above, we interpret pronouns, not as variables, but as noun phrase denotations involving pronoun resolution algorithms: $\texttt{sel} : \gamma e$.

$$[\![pro]\!](P)(c)(\phi) := P(\texttt{sel}(c))(c)(\phi(c))$$

Determiners The system presented in [8] is limited to quantifiers of type $\langle 1 \rangle$.[10,11] Accordingly, we restrict ourselves to the standard universal and existential quantifiers $\forall, \exists : (eo)o$. Determiners *every* and *some* are of type $(et)(et)t$, and denote the following functions.

$$\begin{aligned} [\![every]\!](P)(Q)(c)(\phi) &:= \forall(\lambda x.\neg P(x)(c)(\lambda d.\neg Q(x)(x :: d)(\lambda d.\top)) \wedge \phi(c)) \\ [\![some]\!](P)(Q)(c)(\phi) &:= \exists(\lambda x.P(x)(c)(\lambda d.Q(x)(x :: d)(\phi))) \end{aligned}$$

As explained by De Groote, the negations inside of the lambda term representing the denotation of *every* make the conjunctive meanings of the properties P and Q equivalent to the desired implication. As an example, take $P = [\![man]\!] = \lambda x, c, \phi.\mathbf{man}(x) \wedge \phi(c)$ and take $Q = [\![smile]\!] = \lambda x, c, \phi.\mathbf{smile}(x) \wedge \phi(c)$. Then $[\![every]\!]([\![man]\!])([\![smile]\!])(c)(\phi)$ β-reduces to the following lambda term, taking $A \rightarrow B$ as an abbreviation for $\neg(A \wedge \neg B)$.

$$\forall(\lambda x.\mathbf{man}(x) \rightarrow \mathbf{smile}(x) \wedge \top) \wedge \phi(c)$$

Finally, $\top$ stands for the always true proposition. As it always occurs as part of a conjunction, we simply and systematically replace $A \wedge \top$ everywhere with the equivalent A.

Everything Else All other lexical items are interpreted as the identity function of the appropriate type. While some (such as auxiliaries) should be assigned a more sophistiated denotation in a more sophisticated fragment, others (such as most of the 'functional' lexical items) play no obvious semantic role, and are there purely to express syntactic generalizations.

4 A Fragment

There are two main constructions addressed in this section. First (in §4.1), we show the necessity of de Groote's discourse contexts (or something like them) for the variable naming scheme adopted here. Then (in §4.2), as advertised in the abstract, we illustrate the 'tensed-clause boundedness' of quantifier raising.

We draw lexical items mostly unchanged from [23] (see figure 1). The fragment derives the following sentences, with the indicated scope relations.

4. Some man believed that every woman smiled. $(\exists > \forall)$
5. Some man believed every woman to have smiled. $(\exists > \forall,\ \forall > \exists)$

Although the number of lexical items (especially in the verbal domain) may look at first blush imposing, there are two things to bear in mind. First, most of them are (intended to be) 'closed class' items, meaning that they needn't ever

[10] A quantifier of type $\langle n_1, \ldots, n_k \rangle$ takes k predicates of arities $n_1, \ldots, n_k$ respectively, and returns a truth value.

[11] The extension to quantifiers of type $\langle n \rangle$ for any n is straightforward, but how it should be extended to handle binary quantifiers (of type $\langle 1, 1 \rangle$) is non-obvious.

NAME	FEATURES	PRONUNCIATION	MEANING
that	=s t	"that"	id
-ed	=p +k +q s	"-ed"	id
to	=p t	"to"	id
have	=en p	"have"	id
-en	=v en	"-en"	id
Asp	=v p	"ϵ"	id
Qv	=v +q v	"ϵ"	id
v	=V =d v	"ϵ"	id
AgrO	=V +k V	"ϵ"	id
smile	=d v	"smile"	LIFT(**smile**)
praise	=d V	"praise"	LIFT(**praise**)
believe	=t V	"believe"	LIFT(**believe**)

(a) verbal elements

NAME	FEATURES	PRONUNCIATION	MEANING
dD	=D d -k $\ominus$q	"ϵ"	id
dQ	=D d -k $\ominus\hat{\text{q}}$	"ϵ"	id
every	=n D	"every"	$[\![every]\!]$
some	=n D	"some"	$[\![some]\!]$
man	n	"man"	LIFT(**man**)
woman	n	"woman"	LIFT(**woman**)
pro	D	"he"	$[\![pro]\!]$
j	D	"John"	GQ(**John**)
b	D	"Bill"	GQ(**Bill**)

(b) nominal elements

Fig. 1: Lexical items

$$\begin{aligned}
R(a)(b) &= \mathbf{merge}(a,b) & \mathtt{iv}(s)(v) &= R(v)(s)\\
V(a) &= \mathbf{move}(a) & \mathtt{to}(v) &= R(\mathsf{to})(R(\mathsf{have})(R(\mathsf{-en})(v)))\\
ObjK(v) &= V(R(\mathsf{AgrO})(v)) & \mathtt{pst}(v) &= V(V(R(\mathsf{-ed})(R(\mathsf{Asp})(v))))\\
ObjQ(v) &= V(R(\mathsf{Qv})(v)) & \mathtt{c} &= R(\mathsf{that})\\
Sub(s)(v) &= R(R(\mathsf{v})(v))(s) & \mathtt{d} &= R(\mathsf{dD})\\
\mathtt{tv}(s)(v)(o) &= ObjQ(Sub(s)(ObjK(R(v)(o)))) & \mathtt{q} &= R(\mathsf{dQ})
\end{aligned}$$

Fig. 2: Abbreviations

be added to as more novel words are encountered. Second, these closed class items in fact participate in a very regular way in derivations. We introduce some abbreviations (figure 2), so as to be able to concisely describe derivations of sentences.[12]

The sentence in 6 has the two (semantically equivalent) derivations represented in 7, and interpretation as in 8.

6. Some man smiled.
7. $\mathtt{pst}(\mathtt{iv}(f(\mathsf{some})(\mathsf{man}))(\mathsf{smile}))$, for $f \in \{D, Q\}$
8. $\lambda c, \phi.\ \exists(\lambda x.\ \mathbf{man}(x) \wedge \mathbf{smile}(x) \wedge \phi(c))$

The transitive sentence in 9 has the derivation in 10, which corresponds to the object wide scope reading in 11, and the derivation in 12 which corresponds to the subject wide scope reading in 13.

9. Some woman praised every man.
10. $\mathtt{pst}(\mathtt{tv}(\mathtt{d}(\mathbf{some})(\mathbf{woman}))(\mathbf{praise})(\mathtt{q}(\mathbf{every})(\mathbf{man})))$
11. $\lambda c, \phi.\forall(\lambda x.\mathbf{man}(x) \rightarrow \exists(\lambda y.\mathbf{woman}(y) \wedge \mathbf{praise}(x)(y))) \wedge \phi(c)$
12. $\mathtt{pst}(\mathtt{tv}(\mathtt{q}(\mathbf{some})(\mathbf{woman}))(\mathbf{praise})(\mathtt{q}(\mathbf{every})(\mathbf{man})))$
13. $\lambda c, \phi.\exists(\lambda y.\mathbf{woman}(y) \wedge \forall(\lambda x.\mathbf{man}(x) \rightarrow \mathbf{praise}(x)(y)) \wedge \phi(y :: c))$

[12] The 'abbreviations' in figure 2 are λ-terms over derivation trees. The naming of bound variables is purely mnemonic, as they are all of atomic type.

4.1 Pronouns are not variables

The idea that pronouns are to be rendered as variables in some logical language is widespread in the semantic literature (though see [11, 20] for some alternatives). However in the present system, where possible binders (DPs) bind variables according to the reason for their movement (of which there are only finitely many), sentences like the below prove an insurmountable challenge to this naïve idea.

14. Every boy believed that every man believed that he smiled.

In sentence 14, the pronoun can be bound either by *every boy* or by *every man*. However, if it 'denotes' a variable, it must be one of x_k or x_q, and both of these are bound by the closer DP, *every man*, leaving no possibility for the reading in which every boy smiles.[13] Here we see that treating pronouns as (involving) pronoun resolution algorithms provides a simple way to approach a resolution to this problem.[14]

Sentence 14 has its equivalent derivations in 15, with meaning representation in 16.

15. $\mathtt{pst}(\mathtt{iv}\ (f(\mathsf{every})(\mathsf{boy}))$
$\quad (R(\mathsf{believe})(\mathtt{c}(\mathtt{pst}(\mathtt{iv}\ (g(\mathsf{every})(\mathsf{man}))$
$\quad\quad (R(\mathsf{believe})(\mathtt{pst}(\mathtt{iv}(h(\mathsf{pro})(\mathsf{smile})))))))))$
for $f, g, h \in \{\mathsf{d}, \mathsf{q}\}$
16. $\lambda c, \phi.\forall(\lambda x.\ \mathbf{boy}(x) \rightarrow$
$\quad \mathbf{believe}(\forall(\lambda y.\ \mathbf{man}(y) \rightarrow$
$\quad\quad \mathbf{believe}(\mathbf{smile}(\mathtt{sel}(y :: x :: c)))(y))(x)) \wedge \phi(c)$

Crucially, in every context, the input to the pronoun resolution algorithm includes the (bound) variables x and y, the choice of which would result in the bound reading of the pronoun for *every boy* and *every man* respectively.

4.2 The tensed-clause boundedness of QR

Now we present derivations for sentences 4 and 5 (repeated below as 24 and 17). We begin with 17, which has two surface scope readings which correspond to *de re* (19) and *de dicto* (21) beliefs, and an inverse scope reading (in 23).

17. Some man believed every woman to have smiled.
18. $\mathtt{pst}(\mathtt{tv}(f(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathsf{d}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile})))),$
where $f \in \{\mathsf{d}, \mathsf{q}\}$
19. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \mathbf{believe}(\forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{smile}(y)))(x) \wedge \phi(x :: c))$
20. $\mathtt{pst}(\mathtt{tv}(\mathsf{q}(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathsf{q}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))$

[13] In response to this problem, [23] abandons the idea of [32] that variables are named after movement dependencies, and with it the simply typed lambda calculus.

[14] It is not in itself an answer to this problem, as it introduces an unanalyzed 'unknown' in the form of a pronoun resolution algorithm.

21. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{believe}(\mathbf{smile}(y))(x)) \wedge \phi(x :: c))$
22. $\mathtt{pst}(\mathtt{tv}(\mathtt{d}(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathtt{q}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))$
23. $\lambda c, \phi.\forall(\lambda y.\mathbf{woman} \rightarrow \exists(\lambda x.\mathbf{man}(x) \wedge \mathbf{believe}(\mathbf{smile}(y))(x))) \wedge \phi(c)$

Now we turn to 24. Here the inverse scope reading is not available for the simple reason that a tensed clause (one with the lexical item -ed) forces a Q feature to be checked. Thus the embedded DP *every woman* does not enter into a movement relationship with anything after the matrix subject is present.

24. Some man believed that every woman smiled.
25. $\mathtt{pst}(\mathtt{iv}(f(\mathsf{some})(\mathsf{man}))(R(\mathsf{believe})(\mathtt{c}(\mathtt{pst}(\mathtt{iv}(g(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))))$, for all $f, g \in \{\mathtt{q}, \mathtt{d}\}$
26. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \mathbf{believe}(\forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{smile}(y)))(x) \wedge \phi(x :: c))$

The tensed-clause boundedness of quantifier scope is a fragile and analysis dependent property; a simple 'splitting' of the -ed lexical item into one of the form $\langle -\mathtt{ed}, \mathtt{=p\ +k\ s}\rangle$ and another of the form $\langle \epsilon, \mathtt{=s\ +q\ s}\rangle$ suddenly makes the inverse scope reading in sentence 24 derivable. There are two things to say at this point. First, the 'actual' generalization about scope taking inherent in (this version of) the minimalist grammar framework is that an expression may take scope over only those others dominated by a node in the derivation tree with which it enters into a feature checking relationship. This happens to coincide in our fragment with tensed clauses. Second, the intuitive generalizations this fragment is making are (1) that scope can be checked at the VP and at the S level, and (2) that scope must be checked as soon as possible. If this latter condition is formulated as a transderivational constraint [13], and applied to derivations in the alternative fragment (in which the -ed lexeme is 'split' as per the above remarks), the present fragment can be viewed as the result of 'compiling out' the effect of the constraint on the alternative fragment.

5 Conclusion

We have shown how De Groote's simply typed account of dynamic phenomena can be used in a minimalist grammar. This allows us to maintain Stabler's ideas about variables in movement dependencies, as well as to use nothing but the simply typed lambda calculus to deliver the same meanings as the more standard 'LF' interpretative accounts of semantics in the chomskyian tradition (modulo pronouns). What is doing the work here is the rejection of the pronouns-as-variables view, in favor of a pronouns-as-functions-from-contexts view. This latter seems to be a novel perspective on the the pronouns-as-definite-description view [11] (especially in the light of [9]).

In order to preserve the functional relation between derivations and meanings, we have incorporated information into the feature system (whether or not a licensee feature has a hat diacritic). However, this strategy seems non-ideal, as it results in cases of spurious ambiguity (the worst offender in this paper was example 1, with six equivalent derivations).

Finally, we have noted that there are open questions regarding the best way of incorporating sentential complement embedding verbs and type $\langle 1, 1 \rangle$ quantifiers into the dynamism-via-continuation framework used here [8].

References

1. Abney, S.P.: The English Noun Phrase in its Sentential Aspect. Ph.D. thesis, Massachusetts Institute of Technology (1987)
2. Amblard, M.: Calculs de représentations sémantiques et syntaxe générative: les grammaires minimalistes catégorielles. Ph.D. thesis, Université Bordeaux I (2007)
3. Asher, N., Pogodalla, S.: A montagovian treatment of modal subordination. In: Li, N., Lutz, D. (eds.) Semantics and Linguistic Theory (SALT) 20. pp. 387–405. eLanguage (2011)
4. Barss, A.: Chains and Anaphoric Dependence: On Reconstruction and its Implications. Ph.D. thesis, Massachusetts Institute of Technology (1986)
5. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge, Massachusetts (1995)
6. Cooper, R.: Quantification and Syntactic Theory. D. Reidel, Dordrecht (1983)
7. Cresti, D.: Extraction and reconstruction. Natural Language Semantics 3, 79–122 (1995)
8. de Groote, P.: Towards a montagovian account of dynamics. In: Gibson, M., Howell, J. (eds.) Proceedings of SALT 16. pp. 1–16 (2006)
9. De Groote, P., Lebedeva, E.: Presupposition accommodation as exception handling. In: Fernandez, R., Katagiri, Y., Komatani, K., Lemon, O., Nakano, M. (eds.) The 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue - SIGDIAL 2010. pp. 71–74. Association for Computational Linguistics, Tokyo, Japan (2010)
10. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3: Beyond Words, chap. 3, pp. 125–213. Springer Verlag (1997)
11. Evans, G.: Pronouns. Linguistic Inquiry 11(2), 337–362 (1980)
12. Gärtner, H.M., Michaelis, J.: Some remarks on locality conditions and minimalist grammars. In: Sauerland, U., Gärtner, H.M. (eds.) Interfaces + Recursion = Language?, Studies in Generative Grammar, vol. 89, pp. 161–195. Mouton de Gruyter, Berlin (2007)
13. Graf, T.: A tree transducer model of reference-set computation. UCLA Working Papers in Linguistics 15, Article 4 (2010)
14. Graf, T.: Closure properties of minimalist derivation tree languages. In: Pogodalla, S., Prost, J.P. (eds.) LACL 2011. Lecture Notes in Artificial Intelligence, vol. 6736, pp. 96–111 (2011)
15. Hale, J.T., Stabler, E.P.: Strict deterministic aspects of minimalist grammars. In: Blache, P., Stabler, E.P., Busquets, J., Moot, R. (eds.) Logical Aspects of Computational Linguistics, Lecture Notes in Computer Science, vol. 3492, chap. 11, pp. 162–176. Springer (2005)
16. Harkema, H.: Parsing Minimalist Languages. Ph.D. thesis, University of California, Los Angeles (2001)
17. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell Publishers (1998)

18. Hendriks, H.: Studied Flexibility: Categories and types in syntax and semantics. Ph.D. thesis, Universitaet van Amsterdam (1993)
19. Hornstein, N.: Movement and chains. Syntax 1(2), 99–127 (1998)
20. Jacobson, P.: Towards a variable-free semantics. Linguistics and Philosophy 22(2), 117–184 (1999)
21. Johnson, K.: How far will quantifiers go? In: Martin, R., Michaels, D., Uriagereka, J. (eds.) Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik, chap. 5, pp. 187–210. MIT Press, Cambridge, Massachusetts (2000)
22. Kanazawa, M., Michaelis, J., Salvati, S., Yoshinaka, R.: Well-nestedness properly subsumes strict derivational minimalism. In: Pogodalla, S., Prost, J.P. (eds.) Logical Aspects of Computational Linguistics, LACL 2011. Lecture Notes In Computer Science, vol. 6736, pp. 112–128. Springer, Berlin (2011)
23. Kobele, G.M.: Generating Copies: An investigation into structural identity in language and grammar. Ph.D. thesis, University of California, Los Angeles (2006)
24. Kobele, G.M.: Inverse linking via function composition. Natural Language Semantics 18(2), 183–196 (2010)
25. Kobele, G.M.: Minimalist tree languages are closed under intersection with recognizable tree languages. In: Pogodalla, S., Prost, J.P. (eds.) LACL 2011. Lecture Notes in Artificial Intelligence, vol. 6736, pp. 129–144 (2011)
26. Kobele, G.M., Retoré, C., Salvati, S.: An automata theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds.) Proceedings of the Workshop Model-Theoretic Syntax at 10; ESSLLI '07. Dublin (2007)
27. LeComte, A.: Semantics in minimalist-categorial grammars. In: de Groote, P. (ed.) FG 2008. pp. 41–59. CSLI Press (2008)
28. Michaelis, J.: On Formal Properties of Minimalist Grammars. Ph.D. thesis, Universität Potsdam (2001)
29. Parigot, M.: $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) Logic Programming and Automated Reasoning. Lecture Notes in Computer Science, vol. 624, pp. 190–201. Springer-Verlag, Berlin Heidelberg (1992)
30. Pietrowski, P.M.: Events and Semantic Architecture. Oxford University Press (2005)
31. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)
32. Stabler, E.P.: Computing quantifier scope. In: Szabolcsi, A. (ed.) Ways of Scope Taking, chap. 5, pp. 155–182. Kluwer, Boston (1997)
33. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) Logical Aspects of Computational Linguistics, Lecture Notes in Computer Science, vol. 1328, pp. 68–95. Springer-Verlag, Berlin (1997)