Glue TAG semantics for binary branching syntactic structures

Glue TAG semantics for binary branching syntactic structures

Luke Edward Burke

A thesis presented for the degree of
M.Phil Studies in Philosophy

Department of Philosophy
Birkbeck college, University of London
UK

Abstract

This thesis presents Gl-TAG, a new semantics for a fragment of natural language including simple in/transitive sentences with quantifiers. Gl-TAG utilises *glue semantics*, a proof-theoretic semantics based on linear logic, and TAG, a tree-based syntactic theory. We demonstrate that Gl-TAG is compositional, and bears interesting similarities to other approaches to the semantics of quantifiers.

Chapter 1, rather than discussing the arguments of the thesis as a whole, outlines the global picture of language and semantic theory we adopt, introducing different semantics for quantification, so that Gl-TAG is understood in the proper context.

Chapter 2, the heart of the thesis, introduces Gl-TAG, illustrating its application to quantifier scope ambiguity (Qscope ambiguity) and binding. Ways of constricting quantifier scope where necessary are suggested, but their full development is a topic of future research.

Chapter 3 demonstrates that our semantics is compositional in certain formal senses there distinguished. Our account of quantification bears striking similarities to that proposed in Heim and Kratzer (1998), and also to Cooper storage (Cooper ((1983))); in fact, we can set up a form of Cooper storage within Gl-TAG. We suggest in conclusion that the features in common between frameworks highlight the possible formal similarities between the approaches.

One philosophically interesting aspect of our semantics left aside is that it depends on *proof theoretic* methods; glue semantics combines semantic values both by harnessing the inferential power of linear logic and by exploiting the Curry-Howard isomorphism (CHI) familiar from proof theory (see chapter 2 for a brief explanation of the CHI). The semantic value of a proposition is thus a proof, as some proof theorists have desired (see Martin-Löf (1996). This raises a question for future research; namely, whether Gl-TAG is an inferential semantics in the sense that some philosophers have discussed (Murzi and Steinberger (2015)).

Acknowledgements

I want to thank Paul Elbourne, Ash Asudeh, Wilfrid Hodges, Dag Westerståhl, Herman Hendriks, Wilfred Meyer-Viol, Patrick Elliot, Benat, Neil Barton, Simon Hewitt, Dide Siemmond and my parents.

Contents

Technical preliminaries

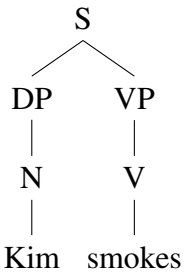
We here provide the necessary technical background for our thesis.

0.0.1 sketches the syntactic theory we presuppose. 0.0.3 discusses the simply typed λ-calculus and type theoretical semantics assumed in much of the literature. Finally, (0.0.6), introduces the algebra needed for chapter 3.

0.0.1 Syntax

Sentences have structure. Parts of sentences contribute to other sentences and act as units, called constituents (Carnie (2010)). In "My cup of tea is without milk", "My cup of tea" is a constituent, whereas "of tea is without" is not. Constituency tests (Adger (2003)), which sort expressions into syntactic categories of the same type, provide evidence that sentences are structured into constituents.

Our semantics, Gl-TAG, applies to constituent-structure trees, such as the following:



The tree consists of a root node, labelled 'S', representing the syntactic category of a sentence, and two branches, one for the two main constituents of the sentence, a determiner phrase and a verb phrase. N and V are *terminal nodes* with expressions attached to them. The categories DP and VP are *immediate constituents* of S, and S is said to *immediately dominate* its immediate constituents. A category immediately dominates its immediate constituents and dominates the constituents of its constituents.

Definition 1 (see Blackburn and Meyer-Viol (1997))

A finite binary branching tree is a tuple $(W, \succ_1, \succ_2, root, \Theta)$ where

W is a finite, non-empty set, the set of tree nodes; $\Theta(\subseteq W)$ contains all and only the trees terminal nodes; and root is the (unique) root node of the tree. $\succ_1$ and $\succ_2$ are binary relations such that:

for all $w, w' \in W$, $w \succ_1 w'$ iff w' is the first daughter of w; and

$w \succ_2 w'$ iff w' is the second daughter of w.

Note that $\succ_1$ and $\succ_2$ are partial functions, for any node in an ordered binary tree has at most one first daughter and a most one second daughter.

Further note that if $w \succ_2 w'$ then there exists a unique w'' such that $w \succ_1 w''$.

Moreover, $w'' \neq w$.

Constituent structures can be mathematically defined: (Hodges 2012: 588-589) provides an axiomatisation of constituent structures, accommodating the role they plays in a wide range of syntactic theories (in fact, any Bare Grammar (Keenan and Stabler 1999), a formalism distilling the common formal core of various syntactic frameworks). Hodges' axiomatisation is based on the intuition that, if f is a constituent of e, some part of f coincides with e, and we can remove this part of e to produce a frame which can be filled in by other expressions:

Definition 2 (Hodges (2012))
A constituent structure is an ordered pair of sets (**E**, **F**), where **E** is the set of grammatical expressions ($e, f, ...$) and F is the set of frames ($F, G, H, ...$), satisfying the following four conditions:

1. F is a set of nonempty partial functions on E. The partial function $F(\xi_1, ..., \xi_n)$ yields a member of E when $e_1, ..., e_n$ are plugged into it, namely $F(e_1, ..., e_n)$.

2. *Nonempty Composition*: If $F(\xi_1, ..., \xi_n)$ and $G(\eta_1, ..., \eta_m)$ are frames, $1 \leqslant i \leqslant n$, and there is an expression

$F(e_1, ..., e_n, G(f_1, ..., f_m), e_{i+1}, ..., e_n)$

then

$F(\xi_1, ..., \xi_{i-1}, G(\eta_1, ..., \eta_m), \xi_{i+1}, ..., \xi_n))$

3. *Nonempty Substitution*: If $F(e_1, ..., e_n$ is an expression, $n > 1$ and $1 \leq i \leq n$, then

$F(\xi_1, ..., \xi_{i-1}, e_i, \xi_{i+1}, ..., \xi_n)$

is a frame.

4. *Identity*: There is a frame 1(ξ) such that for each expression e, $1(e) = e$.

(b) An expression e is a *constituent* of an expression f if f is $G(e)$ for some frame G; e is a *immediate constituent* of an expression f if e is a constituent of f and $e \neq f$.

(c) A set Y of expressions is *cofinal* if every expression of L is a constituent of an expression in Y.

Hodges (2012) defines a syntactic category as follows:

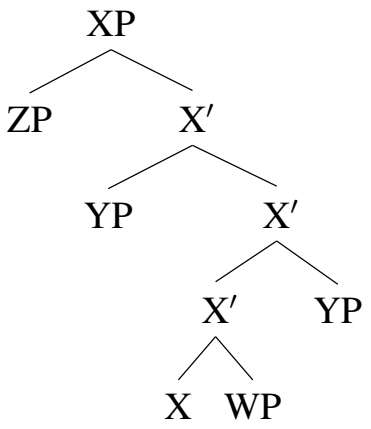
Definition 3 The syntactic category of e and f are identical ($e \sim_X f$) iff for each 1-ary $G \in \mathbb{F}, G(e) \in X$ iff $G(f) \in X$.

The set of expressions of syntactic category X is then the equivalence class $[X]$ with respect to preservation of grammaticality under $\sim_X$.

0.0.2 X-bar theory

X-bar theory (Carnie (2010), Kornai and Pullum (1990)) postulates that every head X of whatever syntactic category projects the same type of basic tree structure:

(1)



Basic schemes
X-bar

- **XP : (ZP) X**
  A phrase (XP) comprises at least a bar-level projection (X) and an optional specifier, ZP. (brackets indicate optionality).
- **X : YP X** or **X: X YP**
  A bar-level projection (X) can contain another bar level projection X and an adjunct, YP. Since YP is a phrase which can again contain another phrase we account for recursive structures in the syntax.
- **X : X (WP)**
  A bar-level projection (X) consists of a head of the same category (X) and a complement, WP.

We note:

1. The specifier is immediately dominated by XP.
2. A phrase that is a sister to a single bar level (N') and a daughter of a single bar level (N') is an adjunct.
3. The sister to X and daughter of X′ is the complement of the head.

X-bar theory allows us to characterise a structural distinction between complements and adjuncts. X bar theory also allows us to rule out given phrase structure rules which do not correspond to structures in natural language, such as (2):

(2) NP $\rightarrow$ V AdjP

X-bar theory is endocentric: it ensures that every XP contains a head of type X, and thus (2) is not possible.

0.0.3 Semantics

The lambda calculus and type theory are often used by semanticists (see Montague (1973), Cresswell (1973), Partee (1975), Williams (1983), Dowty, Wall and Peters (1981), Dowty (1979), Janssen (1983, Heim and Kratzer (1998)) and so we now introduce them.

0.0.4 Basic type theory

In a typed language, well-formed expressions that denote the same kinds of entity are assigned a type, and types are collected into domains called frames (The frame D_a is the set of possible denotations of expressions of type a).

We adopt an extensional type theory (see Henkin (1950) and Gallin (1975) for model-theoretic details). The set of types is defined recursively as follows:

Definition 4 1. e is a type.

2. t is a type.

3. If α and β are types then $\langle \alpha, \beta \rangle$ is a type.

In lieu of 3, the following rule may be adopted: (i) If $\langle a_1, ..., a_n \rangle$ and b are types, then $\langle a_1, ..., a_n, b \rangle$ is a type.)

Definition 4 generates $\langle \alpha, \langle \alpha, \beta \rangle \rangle$, $\langle \langle \alpha, \langle \alpha, \beta \rangle \rangle$, $\langle \alpha \langle \alpha, \beta \rangle \rangle$, as the reader should check.

Central to applications are curried functions:

Definition 5 Given a function $f : (X \times Y) \rightarrow Z$, we have an equivalent series of functions, Curry$(f) : X \rightarrow (Y \rightarrow Z)$. Curry(f) takes an argument of type X and returns a function of type $(Y \rightarrow Z)$, reducing many-place functions to chains of single-argument functions. Thus, $Curry(\langle a_1, ..., a_n, b \rangle) = \langle a_1, \langle a_2, \langle a_3, \langle ... \langle a_n, b \rangle \rangle \rangle \rangle \rangle$.

Currying allows us to produce a denotation for a transitive VP which can then combine with the denotation for the subject.

A standard model for our type theory is a tuple, $<F, I>$, where F is a frame and I is an interpretation function. The interpretation function takes a constant of type τ to a member of the frame of objects of that type, D_τ. Typical domains are the following:

Definition 6 1. $D_e = A \neq \emptyset$: the domain of individuals

2. $D_t = \{0, 1\}$: the domain of truth-values

3. $D_{\langle a,b \rangle}$: the domain of type $\langle a, b \rangle$ functions, (often notated $D_b^{D_a}$)

An intensional model adds a domain, D_s, consisting of the domain of intensions (often the set of indices or points). Sometimes a domain of primitive propositions is assumed (Thomason (1980)).

An interpretation function I for a frame *F* is a function such that: *I* has the set of all constants (for each type, we assume a countably infinite set of variables and a countably infinite set of constants of that type) $I(c_\alpha) \in \mathrm{D}_\alpha$ for each constant c_α of type α:

Definition 7 1. $Var_\tau = \bigcup_\tau Var_\tau$ for any variable of type τ

2. $Con_\tau = \bigcup_\tau Con_\tau$ for any constant of type τ

Definition 8 For every type τ, the set of terms type τ terms the smallest set such that:

1. every type τ constant or variable for any type τ is a term

2. if ϕ and ψ are terms of type τ (formulas), then : $\neg\phi$ and $\phi \wedge \psi$ are formulas;

3. if ϕ is a formula and x is a variable of any type, then $\forall$xϕ is a formula;

4. if A is a term of type $\langle\alpha, \beta\rangle$ and B is a term of type α, then $\langle A, B\rangle$ is a term of type β;

5. if A is a term of type β and x is a variable of type α, then λx. (A) is a term of type $\langle\alpha, \beta\rangle$;

6. if A and B are terms of the same type, then (A = B) is a formula.

The other logical operators have their usual definitions. When parentheses are omitted, association is to the left.

Common types include:

- ${D_t}^{D_e}$ is a characteristic function of a set of individuals. An intransitive verb such as $smokes$ is of this type, for it takes an individual to true iff that individual is a member of a given set. (${D_t}^{D_s}$ is the type of propositions, conceived as (characteristic) functions from intensions to extensions. But since we are sticking to extensional type theory, this need not concern us.)

- ${D_t}^{D_{<e,e>}}$ is a characteristic function of a binary relation between individuals. A transitive verb such as smokes can be understood to be of this type, for it takes an ordered pair of individuals to true iff the first individual stands in a certain relation to the second individual.

- ${D_t}^{D_{\langle e,t\rangle}}$ is the type of generalised quantifiers (functions from predicates of type $\langle e, t\rangle$ to truth values (type t functions). These include quantifiers such as *everyone, someone, etc*. Semantically, for universe E, a so-called type $\langle 1\rangle$ generalised quantifier is a function $f : f \in \mathcal{P}(\mathcal{P}(E))$, or a set of sets of individuals.

- ${D_t}^{D_t}$ is the type of extensional (boolean) truth functions (functions from truth values to truth values). These include the truth-functional connectives.

0.0.5 The lambda calculus

The simply-typed lambda calculus expresses functions of a given type by lambda abstraction:

Definition 9 Syntax: $\lambda - abstraction$:
If u is a variable of type a and α is an expression of type b, then $\lambda u.\alpha$ is an expression of type $\langle \alpha, \beta \rangle$.

Quantifiers and lambda operators bind variables. But quantifiers make formulas from formulas, whereas the λ-operator changes the type of its argument (α in our case) to the type of a function from the type of a variable (u in our case) to the type of its argument, as follows:

Definition 10 Semantics: $\lambda - abstraction)$: If u is a variable of type a, α is an expression of type b, $[\![((\lambda u.\alpha)]\!]^{\mathcal{M},g}$ is that function $f : u_a \mapsto \alpha_b$ such that:

for any object $k \in D_a, f(k) = [\![\alpha]\!]^{\mathcal{M},g}$ where g' is an assigment function just like g except that $g'(u) = k$

Definition 11 Lambda/Beta Conversion (denoted $\Rightarrow_\beta$)) :

$(\lambda x.M)N = M'$

(M' the formula obtained by replacing every occurrence of x in M by N such that any free variable in N is assigned a suitable alphabetic variant)

For definitions of free and bound variables, logical equivalence and axioms for the simply typed λ-calculus, consult Carpenter (1997).

Definition 12 The denotation of a term α, $I(\alpha)^g_{\mathcal{M}}$, with respect to the model $\mathcal{M}$ and assignment function g is written $[\![\alpha]\!]^g_{\mathcal{M}}$, and is defined as follows:

1. $[\![x]\!]^g_{\mathcal{M}} = g(x)$ if $x \in Var$
2. $[\![c]\!]^g_{\mathcal{M}} = [\![c]\!]$ if $c \in Con$
3. $[\![\alpha(\beta)]\!]^g_{\mathcal{M}} = [\![\alpha]\!]^g_{\mathcal{M}}([\![\beta]\!]^g_{\mathcal{M}})$
4. $[\![\lambda x.\alpha]\!]^g_{\mathcal{M}} = f$ such that $f(a) = [\![\alpha]\!]^{g \mapsto a}_{\mathcal{M}}$

The lambda calculus and type theory allow us to assign semantic values to sentences.
Consider the sentence "Some man smokes". We can represent the expressions in the sentence and the way they combine by using the following lambda terms:

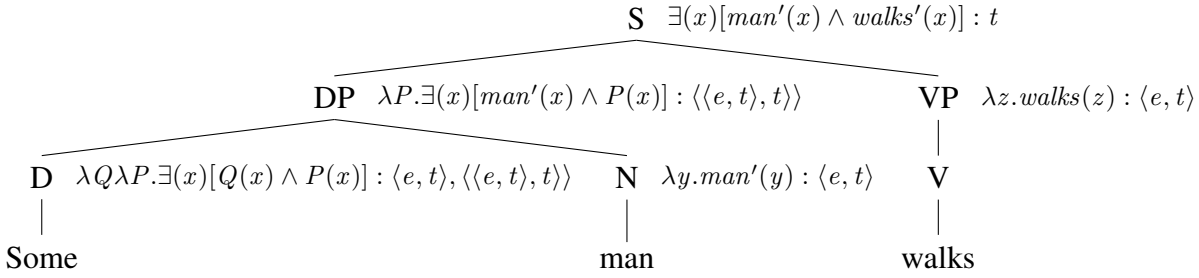
- some: $\lambda Q_{\langle e,t \rangle} \lambda P_{\langle e,t \rangle}.\exists x[Q(x) \wedge P(x)]$
- man: $\lambda y_e.man'(y)$

- some man: $\lambda Q_{\langle e,t\rangle}\lambda P_{\langle e,t\rangle}.\exists x[Q(x) \wedge P(x)](\lambda y.man'(y))$
  (by lambda conversion) $\rightsquigarrow \lambda P_{\langle e,t\rangle}.\exists x[(\lambda y_e.man'(y))(x) \wedge P(x)]$
  (by lambda conversion) $\rightsquigarrow \lambda P_{\langle e,t\rangle}.\exists x[(man'(x) \wedge P(x)]$

- walks: $\lambda z_e.walks'(z)$

- some man walks: $\lambda P_{\langle e,t\rangle}.\exists x[man'(x) \wedge P(x)](\lambda z.walks'(z))$
  (by lambda conversion) $\rightsquigarrow \lambda P_{\langle e,t\rangle}.\exists x[man'(x) \wedge (\lambda z.walks'(z)(x)]$
  (by lambda conversion) $\rightsquigarrow \exists x[man'(x) \wedge walks'(x)]$

We can then assign a denotation to each node of the following simplified tree:



0.0.6 Algebra

Syntax and semantics can be represented as algebras (see Hendriks (2001)). The syntactic algebra consists of a set of expressions, A (the carrier set), and a set of syntactic operations F, and the semantic algebra consists of a set of semantic values B and a set G of semantic operations on them. It is standard (Janssen (1997)) to treat the syntactic (semantic) algebra as many-sorted (syntactic (semantic) rules then apply only to expressions of a certain category) in the following sense:

Definition 13 $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ is a total many-sorted algebra of signature π (a π-algebra) iff

- S is a non-empty set of sorts (intuitively the set of syntactic categories such that A_s is the set of expressions of category s)

- $(A_s)_{s\in S}$ is an indexed family of sets (A_s is the carrier of s)

- Γ is a set of operator indices (total functions that are intuitively the set of syntactic rules)

- π assigns to each $\gamma \in \Gamma$ a pair $\langle\langle s_1, ..., s_n\rangle, s_{n+1}\rangle$, where $n \in \mathbb{N}^+$, $s_1 \in S, ..., s_{n+1} \in S$; and

- $(F_\gamma)_{\gamma\in\Gamma}$ is an indexed family of operators such that if
  $\pi(\gamma) = \langle\langle s_1, ..., s_n\rangle, s_{n+1}\rangle$, then $F_\gamma : A_{s_1} \times ... \times A_{s_n} \mapsto A_{s_{n+1}}$

Some concepts useful to chapter 3 include the following:

Definition 14 A subalgebra of $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ is a π-algebra $\langle (B_s)_{s\in S}, (F'_\gamma)_{\gamma\in\Gamma}\rangle$, where $B_s \subseteq A_s$ for any s. and for $\gamma \in \Gamma$: if $\pi(\gamma) = \langle\langle s_1, ..., s_n\rangle, s_{n+1}\rangle$,, then F'_γ is the restriction of F_γ to $(B_s)_{s\in S}$, that is: $F'_\gamma = F_\gamma \cap ((B_{s_1} \times ... \times B_{s_n}) \mapsto B_{s_n+1})$.

Definition 15 If $(H_s) \subseteq (A_s)$ for all s, then we say that the π algebra includes $(H_s)_{s\in S}$. The smallest subalgebra of A that includes H is the subalgebra generated by H. Suppose $A = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ includes $H = (H_s)_{s\in S}$. Then H is a *generating family* for A iff $\langle [\ H]\ , (F_\gamma)_{\gamma\in\Gamma}\rangle = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$, where $\langle [\ H]\ , (F_\gamma)_{\gamma\in\Gamma}\rangle$ is the subalgebra generated by H.

Def 14 enables us to define the term algebra of A with respect to H ($\langle (T_{A,H,s})_{s\in S}, (F^T_\gamma)_{\gamma\in\Gamma}\rangle$:

Definition 16 For $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ generated by $(H_s)_{s\in S}$, $\langle (T_{A,H,s})_{s\in S}, (F^T_\gamma)_{\gamma\in\Gamma}\rangle$ is a term algebra iff, for all $s \in S$ and all $\gamma \in \Gamma$:

- $(T_{A,H,s})$ is the smallest set such that $\{\lfloor h \rfloor_s \mid h \in H_s\} \subseteq (T_{A,H,s})$, and if $t_1 \in T_{A,H,s_1}), ..., t_n \in T_{A,H,s_n})$ and $\pi(\gamma) = \langle \langle s_1, ..., s_n\rangle, s\rangle$,, then $F^T_\gamma(t_1, ..., t_n) \in (T_{A,H,s})$; and
- $F^T_\gamma(t_1, ..., t_n) = \lfloor \gamma\, t_1, ..., t_n \rfloor_s$

Chapter 1

Introduction

1.1 Language and semantic theory

Philosophers and Linguists have articulated strikingly divergent conceptions of the nature of language and semantic theory (Katz (1980), Chomsky (1986), Stokhof and Van Lambalgen (2011), Scholz and Pullum (2015)). The first half of this chapter therefore briefly surveys the global picture of language and semantic theory we assume, leaving its defence for another occasion.

Our approach to language is constrained by the aim of providing a compositional semantic theory (see chapter 3 for a discussion of compositionality). This narrows down the candidates for a semantic theory, for a technical reason; most technically well developed accounts of compositionality are algebraic; they identify languages with algebras (Hodges (2001), Janssen (1983), Hendriks (2001), Janssen (1997), Montague (1975)) and semantic theories with mathematical structures.

This identification is controversial, perhaps for good reason; it is not *prima facie* obvious how the conception of language as a mathematical structure squares with the naturalistic perspective that Chomsky (1986) and others espouse, according to which language is part of our biological endowment. Nor is it obvious how the algebraic picture squares with the conception of language as governed by mutable social conventions (Lewis (2008), Dummett (1993)) and their communicative rôle.

Since our semantics does not depend on any of these theories, we remain neutral between these conceptions, opting for a weaker position than *identifying* languages and semantic theories with mathematical structures. Rather, we assume that we can *characterise* a language, or core features thereof, using a mathematical structure of a certain kind; to wit, a pair $\langle E, [\![\,]\!] \rangle$ consisting of a partial syntactic algebra, E and an interpretation function $[\![\,]\!]$ taking structural analyses of expressions to their semantic values. A partial syntactic algebra, $\langle\, E, A, \Sigma \,\rangle$ consists of a set of basic expressions, $A \subseteq E$, and a set of partial syntactic operations on those expressions, Σ, that derive the expressions of the language, E.

The distinction between identifying a language with an algebra and characterising a language as an algebra amounts to this: we can use algebras to model salient aspects of natural language without assuming the model is identical to the object modeled. This is in fact the dominant approach towards model-theoretic natural language semantics, as we shall see later. It is also the approach taken by certain formal linguists (Stabler (1997), Graf (2013)), who characterise syntax by means of a set theoretic structure but endorse a broadly chomskyean view of language (Stabler

(2011)).

By assuming that languages can be characterised by mathematical structure we leave space for an appropriate conception of language and semantic theory to emerge from an inspection of our mature theories and the function they serve, as opposed to taking a stand *a priori*. A functional approach of this kind to terms such as "language", "semantic theory" and "semantic value", is encapsulated in Lewis' statement (Lewis (1970)) that "In order to say what a meaning is, we may first ask what a meaning does, and then find something that does that".

For this reason, we spend the following two subsections clarifying the function of the following expressions in our semantic theory, and then give examples of theoretical approaches towards quantification:

- Semantic values and meanings
- Semantic theory

1.1.1 Semantic values and meanings

Much ink has been spilled over what meanings are and whether they are the same thing as mental contents or semantic values (see ft. 1). The medieval Latin linguists operated with a range of different terms to differentiate types of meaning: *sensus, sententia, significatio, consignificatio,* etc (see Hodges (2012)). How can we proceed in the face of such diversity? We will take the position recently articulated by Hodges:

"If these disagreements are just about how to use the word 'meaning' then they aren't worth spending a minute more on... They become significant only when there is an agreed answer to the question 'What facts about language are we aiming to capture with the word 'meaning'? "

Here Hodges echoes the functionalist view about "meaning" and semantic value that we attributed to Lewis above: instead of asking what such terms denote, we try to characterise the explanatory role they play within a particular theory.

As Hodge illustrates, the expression *meaning* has been used in multiply conflicting ways throughout history. Consequently, we use the term *semantic value* rather than the *meaning* for the sake of clarity. The word *semantic value* is arguably a little less less clouded by such diversity of usage. [1]

But note that those theorists that identify a semantic value of a sentence with either a truth value or a function from indices to a truth value assign the same semantic value to extensionally equivalent sentences that differ intuitively in meaning, such as different logical truths. So such theorists also have grounds for distinguishing semantic values and meanings ((Glanzberg 2009:287))

The relevant facts about language we are aiming to capture in our semantic theory with the word *semantic value* are facts concerning the truth conditions of simple intransitive and transitive sentences involving names and quantifiers. We take the following principles regarding truth conditions as basic (for defence, see Cresswell 1982, Zimmermann 2011, Williams 2005):

[1] More substantive grounds have been offered to distinguish semantic values from meanings. A plethora of theorists (Lewis (1981), Rabern (2012), Zimmermann (2006, 2012), Yalcin (2014), Stanley (1997), Ninan (2010), Dummett (1973) argue that meanings cannot be identified with compositional semantic values, largely on the grounds that the two concepts play distinct explanatory roles.

1. *Sentences have truth conditions*: truth conditions are the semantic property of sentences that, given our knowledge of extra-linguistic fact, enable us to track the truth of a given sentence in counterfactual circumstances.
2. Sentences have truth conditions as their semantic values.
3. The truth condition of a sentence is systematically sensitive to its syntactic composition.
4. The semantic value of a sub-sentential expression in a sentence s is its contribution to the truth conditions of s.

Given the functional role described for semantic values, a semantic theory must represent the truth conditions of an arbitrary sentence s in a way compatible with what syntacticians tell us about the structure of s.

1.1.2 Semantic theory: through thick and thin

We can characterise a semantic theory as n-tuple, $\langle\, G, [\![\,]\!], \psi\,\rangle$ consisting of a term algebra G, an interpretation function $[\![\,]\!]$ and a set of rules ψ. This covers a reasonably large class of semantic theories, since we can fit many semantic theories into this mould by having the carrier set G be the set of syntactic structures and ψ be the operations that combine semantic values assigned by $[\![\,]\!]$ to elements of G. An algebraic structure, or mathematical structure of some kind, that characterises the truth conditions of sentences and the way the semantic values of parts of the sentence combine, we call a *thin semantic theory*. A *thick semantic theory* is a theory about the ontology of meaning and its place in the cognitive and social world;[2] it specifies a role for semantic theorising aside from characterising the truth conditions of sentences, articulating its broader goals, scope and methodology.

The danger of confusing thin and thick semantic theories is perceptively diagnosed by Carpenter:

> ...model theory as applied to natural-language semantics has often been misconstrued as making stronger claims than its proponents are comfortable with...the objects employed in models are usually not intended to be provided with a metaphysical interpretation, though many proponents of model theory have done just that. These properties of model-theoretic semantics can be productively compared with the theory of differential equations as applied to the motion of a football. No one would claim that a football computes a differential equation, or even represents one, as it flies through the air....Physicists also realise that most differential equations provide only an approximation of a football's movement, abstracting away from considerations such as turbulence, air temperature, and other details. Nevertheless, differential equations do provide useful approximations... (Carpenter 1997:34)

[2]For a similar distinction see (Murzi and Steinberger 2015: 4): "Semantics concerns itself with the question of which types of semantic values to assign to different categories of expressions, and how, on the basis of these assignments, the semantic values of complex expressions are functionally determined by the semantic values of their constituent expressions. Metasemantics,in contrast, asks two questions: the metaphysical question as to what makes it the case that a given expression has the semantic value it does; and the epistemological question as to what a speaker needs to know to count as understanding the expression."

We agree with Carpenter that, minimally, a semantic theory should expresses useful approximations about the truth conditions of sentences and that it is not necessarily something from which our ontology can be transparently read off. For this reason, we remain neutral on what thick theories are correct. But our choice is also strategic: we wish to avoid getting bogged down in controversial metaphysical questions (viz., about what meanings are) at the outset of our enquiry. We thus concur with Jonnsen that thin semantic theories can be studied apart from thick ones:

> It is worth emphasizing that not all semantic theorizing is concerned with the metaphysical question of what meanings are. Hence, the invocation of a certain kind of semantic value, as part of a semantic theory, is not necessarily a metaphysical proposal... most semantic research done in linguistics and cognitive psychology is concerned with other questions... semantic research in linguistics might instead be concerned with the exploration of the relations between the semantic values of structurally related expressions... Theories... might be seen, by the relevant theoreticians, as instruments of prediction, and possibly explanation, but the theoreticians in question might have no wish to commit themselves to the existence of the entities invoked by their theories. (Jönsson 2008:10)

Our characterisation of semantic theory as the structure $\langle\, G, [\![\,]\!], \psi \,\rangle$ has two advantages. Firstly, it leaves space for a multi-dimensional view of what semantic theory is about. Since a language, $\langle\, E, [\![\,]\!] \,\rangle$, is a subset of a semantic theory, $\langle\, G, [\![\,]\!], \psi \,\rangle$, a semantic theory can be said to be about the language it contains. Likewise, a semantic theory as being about our semantic competence in the sense that the partial term algebra G, the interpretation function $[\![\,]\!]$ and the semantic rules Ψ may be taken to correspond to a mechanism in the speaker by which she is able to interpret and understand arbitrary utterances (Chomsky (1965), Evans (1981)).

Secondly, our characterisation of a semantic theory furnishes criteria for the equivalence of semantic theories and for their correctness. A semantic theory correctly describes the language generated by a certain algebra, if the syntactic algebra of the language and the semantic theory are identical, and the interpretation function of the language and the semantic theory are identical. A stronger criterion of correctness requires in addition that the semantic and syntactic rules of the algebra must correspond to something in the brain of the speaker that explains their linguistic competence.

We can then say that two semantic theories are strongly equivalent if the syntactic algebra and the interpretation function of both the theories are identical and the semantic and syntactic rules of both theories correspond to the same thing in the brain of the speaker that explains their competence. (see Quine (1972) for scepticism about this) [3]

1.2 Approaches to quantification

We now turn to some approaches to quantification. Theories of quantification must account for Qscope ambiguities. Consider (1):

[3] Our criteria for equivalence are similar to those proposed in Zimmermann (2012)

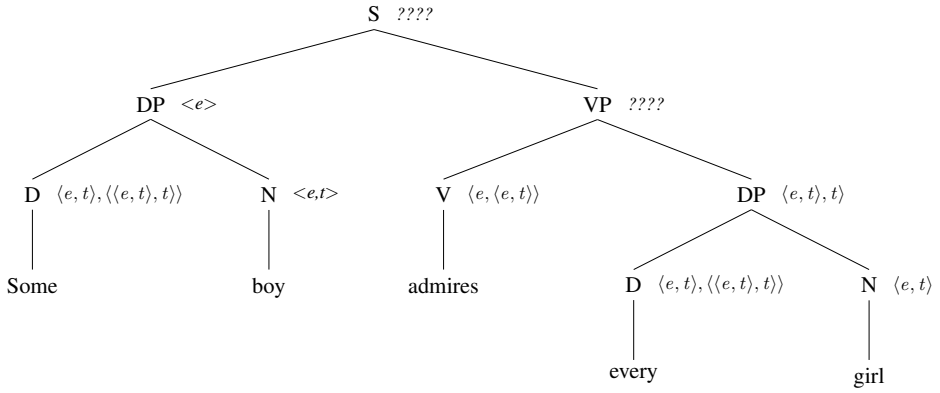
(1) Some boy admires every girl.

This sentence is ambiguous between the following two readings:

(2) a. $\exists x.[Boy'(x) \wedge \forall y[girl'(y) \rightarrow admires(x, y)]]$
b. $\forall y[girl'(y) \rightarrow \exists x[boy'(x) \wedge admires(x, y)]]$

A theory of quantification must generate such ambiguities where necessary, from the meaning of the parts of (1) and their mode of composition. A theory of quantification must also solve the following problem. Consider (3):

(3)



Since transitive verbs are of type $\langle e, \langle \langle e, t \rangle \rangle$ and quantifiers are of type $\langle \langle e, t \rangle, t \rangle$, they cannot combine by functional application, resulting in a composition problem (the problem of the object). We now give a taste of theories of quantification, so that our semantics will be better understood. We distinguish them according to whether they account for the problem of the object and Qscope ambiguities via primarily syntactic or semantic means. [4]

1.2.1 Two syntactic approaches

Montague: Quantifying in

Montague (1975) derives the inverse scope reading of "Some boy admires every girl", by generating two expression, category pairs (t is the category of sentences and T is the category of noun phrases):[5]

(4) α_1, category t: some boy admires her$_{z_n}$

(5) α_2, category T: every girl

A quantification rule (S14) then substitutes the pronoun in α_1 for the quantifier in α_2, to produce "some boy admires every woman".

Corresponding to this S14 is a semantic rule which allows the pronoun to act as a placeholder for the quantifier "every woman" by representing it as a free variable. We then abstract over "some boy admires her$_{z_n}$:

[4] Some of them arguably account for the ambiguity by both syntactic and semantic means.

[5] A rule takes sequences of expressions and their categories and generates an expression category pairs (α_n, category θ)

(6) $\lambda z_3.\exists y.\ boy(y) \wedge\ admires(y, z_3)$

(6) then combines with "Every woman":

(7) $\lambda Q.\forall x.[woman(x) \rightarrow\ Q(x)](\lambda z_3.\exists y.\ boy(y) \wedge\ admires(x, z_3)$

lambda reduction $\rightsquigarrow \forall x.[woman(x) \rightarrow \exists y.\ boy(y) \wedge\ admires(y, x)]$

Montague's approach to Q-scope ambiguities is potentially problematic. For (Janssen 1997) proved that a Montague grammar can generate any recursively enumerable language and its compositionally interpretation, and since the recursively enumerable languages contain all formal grammars (Type-0 grammars in the Chomsky hierarchy), (Janssen 1997)'s result shows that the syntactic rules Montague employs need to be constrained (unless compositionality is a trivial principle satisfied by any formal grammar).

Furthermore, a uniform procedure for dealing with all scopally ambiguous expressions is arguably preferable to a construction specific role for quantifier scope, such as Montague's quantifying in. For these reasons, other theories of quantification may be preferred.

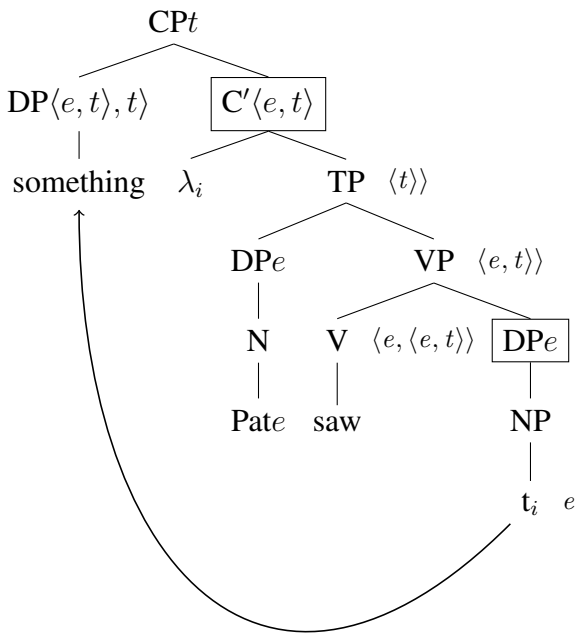
QR

Some generative linguists (Heim and Kratzer (1998)) account for Qscope ambiguities by positing operations mapping a single sentence string to two distinct covert syntactic structures, which receive a distinct interpretation. They posit a rule, QR, which moves a DP from its base position in a syntactic structure to adjoin to a t node higher in the structure, leaving a trace which is co-indexed with the moved quantifier phrase. The problem of the object is resolved by supposing that the trace is of type e (which saturates the verb) and an index is postulated below the moved quantifier which combines with the lower part of the tree by the rule of predicate abstraction, which performs abstraction over the trace:

Composition Rule 6:
Predicate Abstraction Heim and Kratzer (1998)

If γ is a branching node whose daughters are β_i and α, where β is a relative pronoun or 'such', and $i \in \mathbb{N}$ a pronoun or trace, then for any variable assignment a, $[\![\gamma]\!]^a = \lambda\, x.[\![\gamma]\!]^{a^{x/i}}$

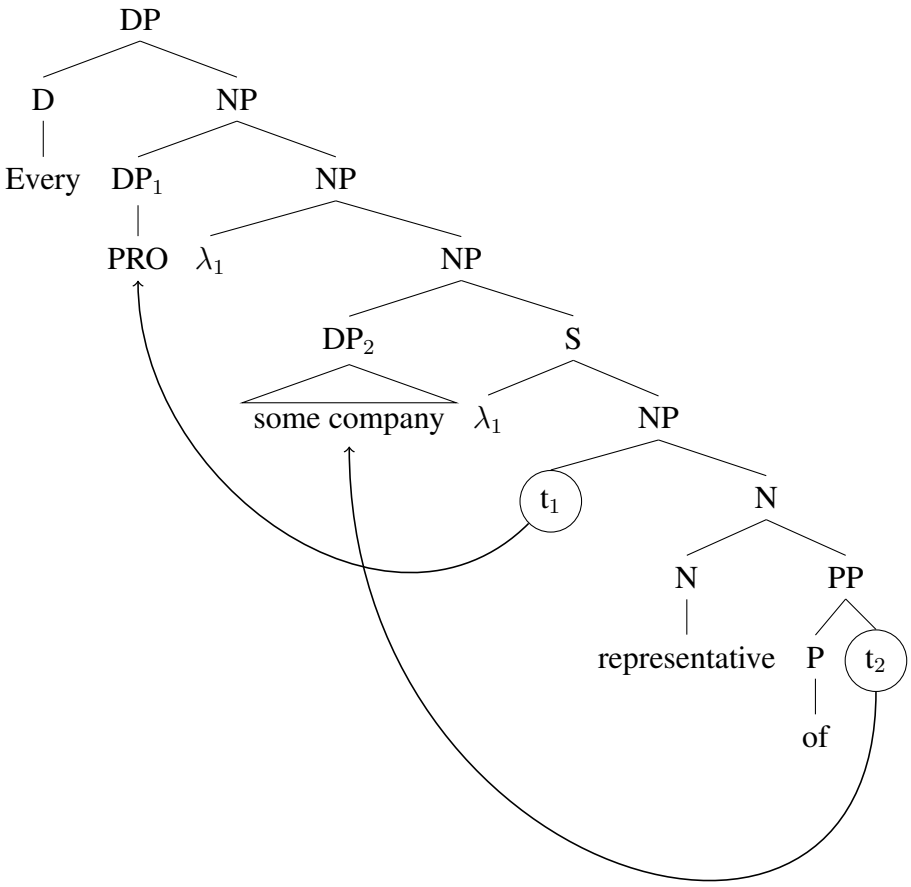
(8)



On this account, Qscope ambiguities arise when quantifiers move to different positions at a distinct level of syntactic representation, LF, at which their surface order is not preserved. Evidence for QR involves parallels between covert QR and overt *wh*-movement (see Aoun and Li 1993), as well as the capacity of QR to contribute to solving problems in syntactic theory (Heim and Kratzer (1998)).

Some theorists have supposed that QR is a particularly natural way to account for quantifier scope (see Von Stechow (1990)). Von Stechow (1990) states that it is "the most transparent method" for dealing with Qscope ambiguities since other methods simply accomplish QR by other means. But Heim and Kratzer (1998) point out that QR sometimes involves complicating the syntactic structure in ways which are contested. Consider the following LF:

(9)



In this LF, the subject position within the NP is filled by a semantically and phonologically null pronoun, PRO, postulated solely in order to derive an interpretable structure. Thus it is not obvious that QR always assigns transparent structures to a sentence.

1.2.2 Two semantic approaches

Cooper storage

Cooper storage is motivated by the idea that "there is no reason from a syntactic point of view to give ... more than one syntactic structure" to Qscope ambiguous sentences. Accordingly, Qscope ambiguities are represented without positing QR. Cooper gives a semantic cast to Montague's syntactic operation of quantifying in, which works as follows. Suppose we have the sentence *John loves some politician*. We first compute *John loves* x_n and store away $< some\ politician, x_n >$. We then retrieve the quantifier and form $\lambda x_n.John\ loves\ x_n$ which combines with the quantifier.

Cooper storage assigns a store to each node of the syntax tree. A store is an n-place sequence containing the lexical entry, alongside pairs (β, i) of NP denotations and indices, present in the core denotation β they are paired with:

Cooper storage

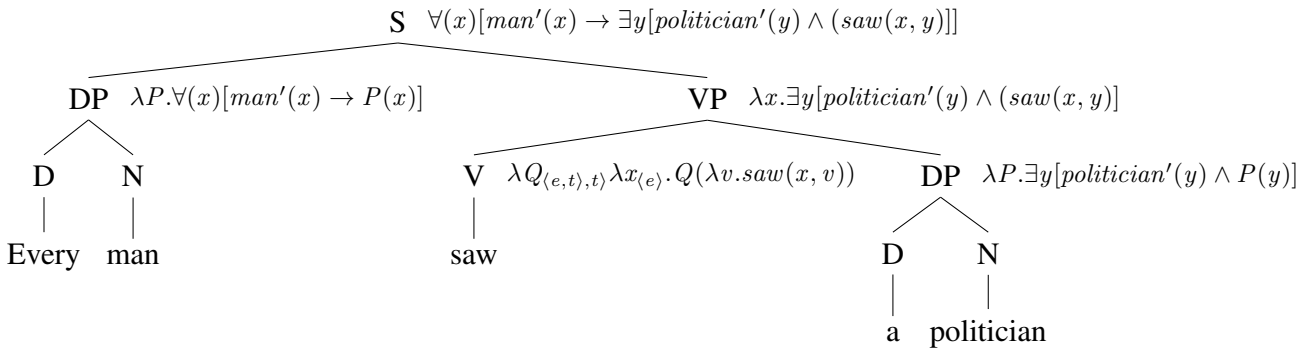
Storage

- If the store $\langle \phi, [\boldsymbol{\beta}, \boldsymbol{j}], \ldots, [\boldsymbol{\beta'}, \boldsymbol{k}] \rangle$ is the denotation of a quantifier phrase Q, then, for i, where i is an index, the store $\langle \lambda P.P(z_i), [\boldsymbol{\phi}, \boldsymbol{i}], [\boldsymbol{\beta}, \boldsymbol{j}], \ldots, [\boldsymbol{\beta'}, \boldsymbol{k}] \rangle$ is also the denotation of Q.

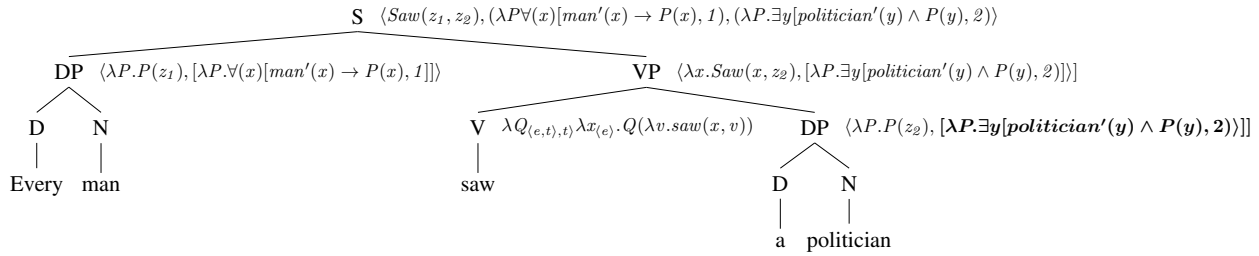
Storage allows us to adopt a new denotation for a quantifier phrase, stowing away the ordinary denotation of a quantifier. The computation of the sentence can then proceed by utilising a place holder, $\langle \lambda P.P(z_i)$ instead of the quantifier denotation.

The first of the following two trees shows the calculation of the ordinary semantic value of each node of the tree, where storage has not applied (the verb has been type raised to take a quantifier as direct object). The second of the following trees shows the storage rule being applied to the two quantifiers in the sentence

(10)



(11)



To proceed, we apply retrieval to the root of (11):

Cooper storage
Retrieval

- Let $[\boldsymbol{\sigma_1}]$ and $[\boldsymbol{\sigma_2}]$ be sequences of binding operators (which may be empty), and $[\beta, i]$ be given stored quantifier. If the denotation of the root node is the store $\langle \phi, [\boldsymbol{\sigma_1}], [\boldsymbol{\beta, i}], [\boldsymbol{\sigma_2}] \rangle$, then the denotation of the root node is also $\langle \beta(\lambda z_i.\phi), [\boldsymbol{\sigma_1}], [\boldsymbol{\sigma_2}] \rangle$.

Retrieving "Every man" first derives the inverse scope reading (Retrieving "a politician" first derives the surface scope reading):

1. $\langle Saw(z_1, z_2), [\lambda P \forall(x)[man'(x) \rightarrow P(x), 1], [\boldsymbol{\lambda P.\exists y[politician'(y) \wedge P(y), 2}]\rangle$
2. by retrieval $\rightsquigarrow \langle \lambda P \forall(x)[man'(x) \rightarrow P(x)(\lambda z_1.Saw(z_1, z_2), [\boldsymbol{\lambda P.\exists y[politician'(y) \wedge P(y), 2)}]\rangle$
3. by lambda conversion $\rightsquigarrow \langle \lambda P \forall(x)[man'(x) \rightarrow (\lambda z_1.Saw(z_1, z_2)(x),$ $[\boldsymbol{\lambda P.\exists y[politician'(y) \wedge P(y), 2)}]\rangle$
4. by lambda conversion $\rightsquigarrow \langle \forall(x)[man'(x) \rightarrow Saw(x, z_2), [\boldsymbol{\lambda P.\exists y[politician'(y) \wedge P(y), 2)}]\rangle$
5. by retrieval $\rightsquigarrow \langle \lambda P.\exists y[politician'(y) \wedge P(y)(\lambda z_2 \forall(x)[man'(x) \rightarrow Saw(x, z_2)\rangle$
6. by lambda conversion $\rightsquigarrow \langle \exists y[politician'(y) \wedge \lambda z_2 \forall(x)[man'(x) \rightarrow Saw(x, z_2)(y)\rangle$
7. by lambda conversion $\rightsquigarrow \langle \exists y[politician'(y) \wedge \forall(x)[man'(x) \rightarrow Saw(x, y)\rangle$

Cooper storage encounters problems, however, when dealing with nested DPs. Consider the following sentence:

(12) John saw every owner of an espresso machine.

The sentence node before retrieval is the following:

(13) $\langle Saw(John, z_3), [\lambda P.\forall x[owner(x) \wedge of(x, z_4) \rightarrow P(x)], 3],$ $[\lambda Q.\exists y[espressomachine(y) \wedge Q(y)], 4]\rangle$

If we retrieve the universal first and then the existential and perform lambda conversion we get:

(14) $\lambda P.\forall x[owner(x) \wedge of(x, z_4) \rightarrow Saw(John, x), [\lambda Q.\exists y[espressomachine(y) \wedge Q(y)], 4]\rangle$

$\exists y[espressomachine(y) \wedge (\forall x[owner(x) \wedge of(x, y) \rightarrow Saw(John, x)\rangle$

If we retrieve the existential first and then the universal and perform lambda conversion we get:

(15) $\exists y[espressomachine(y) \wedge Saw(John, z_3), [\lambda P.\forall x[owner(x) \wedge of(x, z_4) \rightarrow P(x), 3]$

$\lambda P.\forall x[owner(x) \wedge of(x, z_4) \rightarrow P(x)(\lambda z_3.\exists y[espressomachine(y) \wedge Saw(John, z_3)$

But then we end up with a free variable in the antecedent of the formula:

(16) $\forall x[owner(x) \wedge of(x, z_4) \rightarrow \exists y[espressomachine(y) \wedge Saw(John, x)$

Keller (1988) avoids this problem, by redefining storage and retrieval, so that stores can be nested within one another:

Keller storage

K.Storage

- If the store $\langle \phi, \sigma \rangle$ is the denotation of a quantifier phrase Q, then, where i is an index, the store $\langle \lambda P.P(z_i), (\phi, \sigma, i) \rangle$ is also the denotation of Q.

Keller retrieval

K.Retrieval

- Let $[\boldsymbol{\sigma}]$, $[\boldsymbol{\sigma_1}]$ and $[\boldsymbol{\sigma_2}]$ be sequences of binding operators (which may be empty), and $[\beta, i]$ be some chosen stored quantifier we wish to retrieve. If the denotation of the root node is the nested store $\langle \phi$, $[\boldsymbol{\sigma_1}]$, $[(\boldsymbol{\beta}, \boldsymbol{\sigma}), \boldsymbol{i}]$, $[\boldsymbol{\sigma_2}]$ $\rangle$, then the denotation of the root node is also $\langle \beta(\lambda z_i.\phi)$, $[\boldsymbol{\sigma_1}]$, $[\boldsymbol{\sigma}]$, $[\boldsymbol{\sigma_2}]$ $\rangle$, where β takes $(\lambda z_i.\phi)$ as its argument.

Consider (13), repeated here:

(17) $\langle Saw(John, z_3), [\lambda P.\forall x[owner(x) \wedge of(x, z_4) \rightarrow P(x)], 3],$
$[\lambda Q.\exists y[espressomachine(y) \wedge Q(y)], 4]\rangle$

By Keller retrieval we can put "owner of an espresso machine" in a store:

1. $\langle Saw(John, z_3), [\lambda P.\forall x[owner(x) \wedge \exists y\ expresso\ machine(y) \wedge of(x, y) \rightarrow P(x), 3]]\rangle$
2. By retrieval $\rightsquigarrow \langle \forall x[owner(x) \wedge \exists y\ expresso\ machine(y) \wedge of(x, y) \rightarrow Saw(John, x)]]\rangle$

To get the inverse scope reading we store "an espresso machine" separately:

1. $\langle Saw(John, z_3), [\lambda P.\forall x[owner(x) \rightarrow P(x)], [\lambda Q.\exists y\ expresso\ machine(y) \wedge Q(y)]3], 4]$
2. We then retrieve the universal followed by the existential (I have admitted the steps of lambda abstraction and reduction):

$\langle \forall x[owner(x) \wedge of(x, z_4) \rightarrow Saw(John, x)], [\lambda Q.\exists y\ expresso\ machine(y) \wedge Q(y), 4]$

$\langle \exists y\ expresso\ machine(y) \wedge (\lambda z_4 \forall x[owner(x) \wedge of(x, z_4) \rightarrow Saw(John, x)]$

3. by lambda reduction: $\rightsquigarrow \langle [\exists y\ expresso\ machine(y) \wedge (\forall x[owner(x) \wedge of(x, y) \rightarrow Saw(John, x)]]$

But even the revision does not solve all the problems (see Blackburn and Bos (2005). In particular, Cooper storage overgenerates scope readings (Fox and Lappin (2008)). A simple sentence such as the following turns out to have 120 (5!) readings, if we allow any arbitrary scoping of its quantifiers:

(18) Some representatives of every department in most companies saw a few samples of each product.

This means that we have to specify some mechanism by which certain scope readings are rendered unavailable. Cooper proposed a way of making his mechanism sensitive to islands, by requiring that all stored quantifiers in the interpretation of an island node be retrieved before it can semantically with another constituent. But islands are not the only factor that prevents free scoping of quantifiers, thus Cooper needs to make his mechanism sensitive to other constraints.

Hendriks: flexible types

Hendriks (1993) accounts for Qscope ambiguity by assigning a richer denotation to expressions in the grammar, associating each syntactic category with the possibly infinite set of types to which it can shift. The following rules are used to shift types:

Flexible types
n-value raising:

- For a sequence of types $\vec{a}$, if α is of type $\langle \vec{a}, b_i \rangle$ it is also of type $\langle \vec{a} \langle \langle b, d \rangle, d \rangle \rangle$, for possibly empty $\vec{a}$
- Semantics: $\alpha_{\langle \vec{a}, b \rangle} \mapsto \lambda \vec{x}_{\alpha} \lambda P_{\langle b,d \rangle}.[P(\alpha(\vec{x}))]$
- When $\vec{a} = \emptyset$ and $b = e, d = t$, this derives a $\langle e, t \rangle, t \rangle \rangle$ denotation for a proper name of type e

Flexible types

n-argument raising:

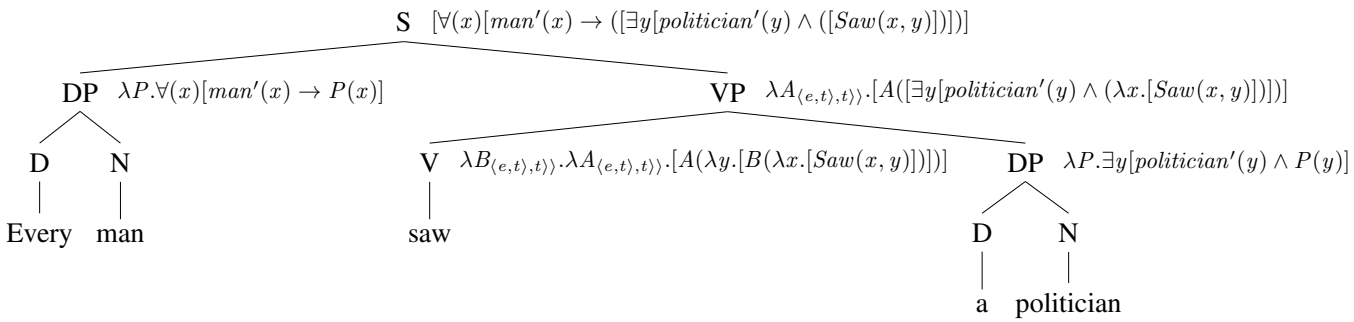
- If α is of type $\langle \vec{a}, \langle b, \langle \vec{c}, d\rangle\rangle\rangle$, then it is of type $\langle \vec{a}, \langle\langle\langle b, d\rangle, d\rangle, \langle \vec{c}, d\rangle\rangle\rangle$
- Semantics: $\alpha_{\langle \vec{a}, \langle b, \langle \vec{c}, d\rangle\rangle\rangle} \mapsto \lambda \vec{x}_{\alpha} \lambda P_{\langle\langle b,d\rangle,d\rangle} \lambda \vec{z}_{c} [P(\lambda y_b \alpha(\vec{x})(y)(\vec{z}))]$
- n-argument raising shifts a transitive verb of type $\langle e, \langle e, t\rangle\rangle$ to a verb taking a quantifier in direct object position of type $\langle \langle e, t\rangle, t\rangle\rangle, \langle e, t\rangle$.

By applying argument raising twice to a transitive verb, depending on the order of these operations, the verb denotes either of the following relations between generalised quantifiers (which derive the surface scope and inverse scope readings of Qscope sentences):

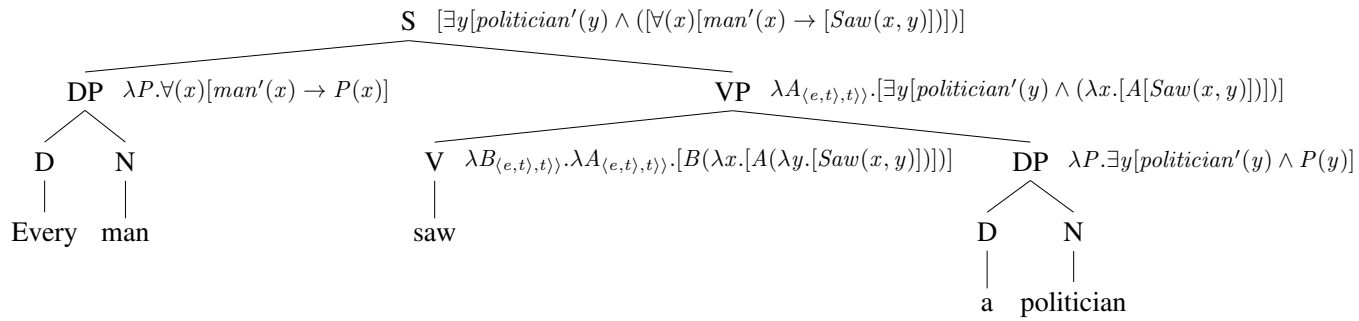
- S $>$ O : $\lambda x_{\langle e\rangle} \lambda y_{\langle e\rangle}.[saw(x,y)] \mapsto \lambda B_{\langle\langle e,t\rangle,t\rangle\rangle}.\lambda A_{\langle\langle e,t\rangle,t\rangle\rangle}.[A(\lambda y.[B(\lambda x.[Saw(x,y)])])]$
- O $>$ A : $\lambda x_{\langle e\rangle} \lambda y_{\langle e\rangle}.[saw(x,y)] \mapsto \lambda B_{\langle\langle e,t\rangle,t\rangle\rangle}.\lambda A_{\langle\langle e,t\rangle,t\rangle\rangle}.[B(\lambda x.[A(\lambda y.[Saw(x,y)])])]$

A sentence such as "Every man saw a politician" then has the following semantics:

(19)



(20)



Let us compute the surface scope reading of the above sentence. First, apply argument raising to the first argument of a transitive relation to derive:

(21) $[1AR](saw) = \lambda Q_{1\langle e,t\rangle,t\rangle}.\lambda x_{\langle e\rangle}.[Q_1(\lambda y.Saw(y,x))$

Then apply argument raising to the second argument of (21) to derive:

(22) $[2AR][1AR](saw) = \lambda Q_{1\langle e,t\rangle,t\rangle}\lambda Q_{2\langle e,t\rangle,t\rangle}.Q_2(\lambda x_{\langle e\rangle}[1AR](saw)(Q_1)(x)$
$= \lambda Q_{1\langle e,t\rangle,t\rangle}\lambda Q_{2\langle e,t\rangle,t\rangle}.Q_2(\lambda x_{\langle e\rangle}Q_1(\lambda y.saw(y)(x)))$

(22) takes "a politician" as argument:

(23) $[2AR][1AR](saw)(a\ politician) = \lambda Q_{2\langle e,t\rangle,t\rangle}.Q_2(\lambda x_{\langle e\rangle}\lambda P_{\langle e,t\rangle}\exists z[Pol(z) \wedge P(z)](\lambda y.Saw(y)(x)))$
$= \lambda Q_{2\langle e,t\rangle,t\rangle}.Q_2(\lambda x_{\langle e\rangle}\exists z[Pol(z) \wedge saw(z)(x)))$

(23) takes "Every man" as argument:

(24) $[2AR][1AR](saw)(a\ politician)(every\ man)$
$= \lambda Q_{2\langle e,t\rangle,t\rangle}.Q_2(\lambda x_{\langle e\rangle}\exists z[W(z) \wedge saw(z)(x)])(\lambda P_{\langle e,t\rangle}\forall v[M(v) \rightarrow P(v)]$
$= \forall[M(v) \rightarrow \exists z Pol(z) \wedge saw(z)(v)]]$

To derive the inverse scope reading we first apply argument raising to the second argument of a transitive relation, and the computation is as above.

Hendriks' approach, insofar as it is a theory of our semantic competence, is problematic, since it requires a strong empirical hypothesis for which we need evidence: that speakers know the rules which generate an infinite set of semantic values for any expression and that positing such rules is desirable for a semantic theory. Note that this is a distinct empirical hypothesis from the assumption made by syntacticians that speakers have knowledge of a grammar, generating an infinity of sentences. Hendriks' requires evidence for his particular hypothesis. A further problem with Hendriks' approach is that it requires a set-valued function between the syntax and the semantics if it is to be compositional and retain its distinctive, semantic explanation of Qscope ambiguity. Set-

valued functions of this kind have been shown to be problematic in some cases (see Westerstahl (2007), so we might wish to see if Hendriks' approach can be compositional under a relational version of compositionality, such as Westerstahl (2007). It can't. For Westerståhl's account requires, contrary to Hendriks' approach, that a semantics is bounded, in that every expression has finitely many semantic values (the requirement might be based on the lack of empirical evidence that speakers know general procedures that generate the infinite set of an expressions types). A semantics R is *bounded* if for each operation α of the syntax the number of semantic values of a term of the form $\alpha(q_1, ..., q_n)$, where any q_i has m_i as some meaning, is less than or equal to some finite number k:

Definition 17 $\left| \bigcup_{i=1} \left\{ R_{\alpha(q_1,...,q_n)} : \bigwedge_{1 \leq i \leq n} R(q_i, m_i) \right\} \right| \leq k$

Westerståhl (p.c) suggests Hendriks' semantics could be made bounded if the infinitely many types of an expression corresponded to infinitely many distinct syntactic terms. But then Hendrik's explanation of Qscope ambiguity would no longer be purely semantic; it would account for such ambiguities by positing distinct terms. Furthermore, it would entail that the speaker has knowledge of an infinitely many syntactic terms, which may be problematic given our finite capacities. Even if Hendriks' can account for Qscope ambiguities by purely semantic means, Hendriks (p.c) has conceded that, given the brain of the language user is finite, there may be a problem with his semantics satisfying compositionality, since the homomorphic algebras involved in defining a compositional semantics must be finitely generated, which is not the case in his account.

Von Stechow (2009) argues that Hendriks' approach is a kind of QR by other means:

"... a closer look at the meta-language in which these type shifters is formulated shows that QR is hidden in the formulation of the rules."

We agree that something achieves the effects of QR in the semantic metalanguage. But Von Stechow is being tendentious, and is making a category mistake; QR is no more a semantic operation than type shifting is a syntactic rule. Most theories of quantifier scope ambiguity have the same effect of QR; they assign some predicate abstract to a sentence which a quantifier scopes above and takes as argument. This feature is present in Montague's quantifying in, in Cooper semantics and in most other approaches. Thus Von Stechow could equally have said that the rule of quantifying in, or storage is hidden in the formation of the rules.

1.2.3 Conclusion

We have described our approach to language and to semantic theory, according to which languages and semantic theories can be characterised as algebras and the semantic values of sentences are their truth conditions. We have outlined various theories of quantification and their explanations of Qscope ambiguity. In the next chapter we outline our semantics and in the final chapter we show that it is compositional.

Chapter 2

A taste of glue

2.1 Introduction

This chapter outlines Gl-TAG, a novel approach to the syntax-semantics interface based on semantic trees derived by the operations of a TAG grammar. Parts of the constituent structure of a sentence are systematically mapped to semantic trees, which are parse trees corresponding to a glue semantics proof. The glue semantics proof then receives a model theoretic interpretation. Gl-TAG allows us to pair two semantic interpretations with one syntactic structure. Sentences that give rise to quantifier scope ambiguities (Qscope ambiguities) are then analysed as syntactically unambiguous, but giving rise to a set of Gl-TAG trees, which receives their model theoretic interpretations. Gl-TAG trees bear resemblances to the LFs postulated in Heim and Kratzer (1998), and to Cooper Storage (Cooper ((1983)). Gl-TAG trees suggest different methods of dealing with Qscope ambiguities share some features.

In 2.1.1 we introduce linear logic and the Curry Howard Isomorphism, which glue semantics is based on. In 2.2.1 we give a taste of glue proofs. We then introduce our semantics gradually; first discussing parse trees (2.3), and then (2.3.2) TAG,before moving on to discuss simple in/transitive sentences (2.4.1), quantifiers (2.4.2) and Qscope ambiguities (2.4.3) and binding (2.4.4). Finally, we suggest ways in which the scope of quantifiers can be restricted (2.4.5).

2.1.1 What is linear logic?

We will be using the very weak implicational fragment of linear logic (the only connective is $\multimap$), a logic invented by the proof theorist Girard (Girard (1987)). Linear logic is a substructural logic (see Restall (2002), Restall (2006)) since it abandons some of the structural rules of the sequent calculus: contraction and weakening. Suppose we have a sequent $\Gamma \vdash \Delta$ (for formulas Γ, Δ and entailment relation $\vdash$). Weakening (on the L_W/R_W) weakens the antecedent (succedent) of a sequent by introducing an extra formula, D:

$$\frac{\Gamma \vdash \Delta}{D, \Gamma \vdash \Delta}\,\mathrm{L}_W$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, D}\mathrm{R}_W$$

Contraction discards repetitions of formulas in the antecedent (succedent) of a given sequent:

$$\frac{D, D, \Gamma \vdash \Delta}{D, \Gamma \vdash \Delta} \mathrm{L}_C$$

$$\frac{\Gamma \vdash \Delta, D, D}{\Gamma \vdash \Delta, D} \mathrm{R}_C$$

The abandonment of contraction entails that the relata of logical consequence are not sets of premises, but rather multisets. (If the relata were sets then contraction would be valid since $\{A, A\} = \{A\}$.)

Consider a classically valid inference:

(1) $A \to (B \to C), A \to B \vdash A \to C$

The sequent calculus proof of this uses contraction and is therefore not derivable in linear logic (unless special modalities are used):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{Axiom} \quad \dfrac{\dfrac{}{A \vdash A}\,\text{Axiom} \quad \dfrac{\dfrac{}{B \vdash B}\,\text{Axiom} \quad \dfrac{}{C \vdash C}\,\text{Axiom}}{B \to C, B \vdash C}\,{\to_L}}{A \to (B \to C), A, B \vdash C}\,{\to_L}}{A \to (B \to C), B, A \vdash C}\,\text{Permutation}_L}{A \to (B \to C), A \to B, A, A \vdash C}\,{\to_L}}{A \to (B \to C), A \to B, A \vdash C}\,\text{Contraction}_L}{A \to (B \to C), A \to B \vdash A \to C}\,{\to_R}$$

Nor can (1) be proven in linear logic, since contraction corresponds in natural deduction to discharging the same hypothesis more than once, which is not allowed in linear logic:

$$\dfrac{\dfrac{\dfrac{A \to (B \to C), A \to B \vdash A \to C}{A \to (B \to C) \quad [A]^1}\,{\to_E} \quad \dfrac{A \to B \quad [A]^1}{B}\,{\to_E}}{B \to C}\,{\to_E}}{\dfrac{C}{A \to C}\,{\to_I}^1}$$

Linear logic abandons these structural rules due to the possible benefits (Girard (1987)) of conceiving of premises as non-renewable resources; a premiss is therefore modeled as a finite

resource which must be used once. Consequently, a given hypothesis cannot be endlessly reused in proof, and has a limited shelf life.

An example illustrates a potential benefit of the resource conception. Suppose £1 will purchase exactly one coffee, or exactly one copy of the newspaper. Let A be 'I have £1', B be 'I can buy a coffee' and C be 'I can buy the newspaper'. Then, where $\rightarrow$ denotes "if...then", we have $A \rightarrow B$ and $A \rightarrow C$. But $A \rightarrow (B \wedge C)$ doesn't follow in this situation: I cannot both buy a newspaper and a coffee. Nonetheless, $(A \rightarrow B, A \rightarrow C \vdash A \rightarrow B \wedge C)$ is classically correct. In order for $(A \rightarrow B, A \rightarrow C \vdash A \rightarrow (B \wedge C))$ to hold we have to discharge A twice, which is not permissible in linear logic.

The resource-theoretic conception of logic has been argued to have advantages over other proof theoretic rivals (Asudeh (2012), Crouch and van Genabith (2000). For a proof theoretic semantics that allows hypotheses to be freely assumed and discharged risks allowing certain semantic values to be used more than once in the sentence. Consider the sentence, "I kissed the philosopher's alleged son". We want the contribution of "alleged" to be made only once, in modifying "son", and we do not want "alleged" to be able to modify "philosopher" thereby deriving "I kissed the alleged philosopher's alleged son." By restricting re-use of hypotheses, glue avoids this problem.

2.2 Glue: an introduction

Glue originated in the semantics-syntax interface for LGF syntax (Dalrymple (1999)). It has since been applied to a number of syntactic frameworks, including HPSG (Asudeh and Crouch (2001b)), Categorial Grammar (Asudeh and Crouch (2001a)), TAG and Minimalism (Gotham (2015)), and we here apply glue to the simplified binary syntactic structures familiar from Minimalism (Adger (2003)).

In glue, lexical entries for constituents of the sentence are two part *meaning constructors*, which are of the form $\mathcal{M} : G$, where $\mathcal{M}$ is some term of the simply-typed λ-calculus that represents the semantic value of the given lexical item and G is a term of linear logic (representing the type of the lambda function). The Curry-Howard isomorphism (CHI) proves terms of whatever type in the simply typed λ-calculus can be assigned a formula of linear logic which is valid iff the term is.[1] For example, if we have a lambda term of type $\langle e, t\rangle$, we can pair this with the linear logic propositional formula $A \multimap B$ and if we have a lambda term of type $\langle e, t\rangle, \langle e, t\rangle, t\rangle\rangle$ we can pair this with the linear logic propositional formula $(A \rightarrow B) \rightarrow ((A \rightarrow B) \rightarrow B))$. Consequently, lexical entries for parts of the sentence can be treated as premises for proofs, thus harnessing the proof theoretical features of natural deduction to give denotations to sentences.

Glue proofs exploit two particular properties of the CHI. Firstly, they exploit the correspondence between implication elimination and function application which the CHI establishes. The following diagram shows the correspondence in schematic terms:

(2.1) Function application = Implication Elimination

[1] In computer science, the CHI is used to show that formulas can be correlated with types in such a way that each proof of a formula corresponds to a program having a certain type.

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ a : A & f : A \multimap B\end{array}}{f(a) : B}\ \multimap\mathcal{E}$$

The diagram shows a function, f (paired with a linear logic conditional) applied to the meaning constructor $a : A$.

Suppose we have the meaning constructor $j' : A$ denoting an individual and the meaning constructor $\lambda x.P(x) : A \multimap B$, denoting the characteristic function of set P. Then, by 2.2 we have:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ j' : A & \lambda x.P(x) : A \multimap B\end{array}}{\lambda x.P(x)(j') : B}\ \multimap\mathcal{E}$$

By lambda reduction ($\Rightarrow_\beta$) we have:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ j' : A & \lambda x.P(x) : A \multimap B\end{array}}{P(j') : B}\ \multimap\mathcal{E}, \Rightarrow_\beta$$

The second property of the CHI glue proofs exploit is the correspondence it establishes between implication introduction and lambda abstraction. The CHI establishes that implication introduction is equivalent to a variable being assumed and then abstracted over by lambda abstraction when the assumption is discharged. The following diagram shows the correspondence schematic terms:

(2.2) Lambda abstraction : Implication Introduction

$$\dfrac{\begin{array}{c}[x : A]^1 \\ \vdots \\ f : B\end{array}}{\lambda x.f : A \multimap B}\ \multimap\mathcal{I},1$$

To give a flavour of how glue proofs work, we give glue proofs, of the sort glue semanticists give, for the following simple sentences

(2) a. Kim sleeps.
b. Kim hugs Robin.
c. Kim hugs everyone.
d. Everybody hugs someone.

Pay attention to the meaning constructors in these proofs, and remember that instances of function application and abstraction in lambda terms correspond to implication elimination and introduction in the linear logic formulas to the right of the colon.

2.2.1 Example I: Transitive with Proper Name Arguments

(2.3) Kim hugs Robin.

Instantiated lexicon/premises for (2.3):

Kim	e	$kim : A$
Robin	e	$robin : B$
hugged	$e \rightarrow (e \rightarrow t)$	$\lambda y.\lambda x.hug(x, y) : B \multimap (A \multimap C)$

Full Proof for (2.3):

$$\dfrac{\dfrac{\dfrac{\dfrac{\textbf{hug}\quad \lambda y.\lambda x.hug(x,y) : B \multimap A \multimap C \qquad \textbf{Robin}\quad robin : B}{(\lambda y.\lambda x.hug(x,y))(robin) : A \multimap C}\multimap\varepsilon}{\lambda x.hug(x, robin) : A \multimap C}\Rightarrow_\beta \qquad \textbf{Kim}\quad kim : A}{(\lambda x.hug(x, robin))(kim) : C}\multimap\varepsilon}{hug(kim, robin) : C}\Rightarrow_\beta$$

2.2.2 Example II: Transitive with Quantifier Object

(2.4) Kim hugged everybody.

Instantiated lexicon/premises for (2.4):

Kim	e	$kim : B$
hugged	$e \rightarrow (e \rightarrow t)$	$\lambda y.\lambda x.hug(x, y) : A \multimap (B \multimap C)$
everybody	$(e \rightarrow t) \rightarrow t$	$\lambda P.\forall y.[person(y) \rightarrow P(y)] : \forall X.(A \multimap X) \multimap X$

Proof for (2.4), abridged for $\Rightarrow_\beta$:

$$\dfrac{\dfrac{\lambda P.\forall y.[person(y) \rightarrow P(y)] : \forall X.(A \multimap X) \multimap X}{\lambda P.\forall y.[person(y) \rightarrow P(y)] : (A \multimap C) \multimap C}\forall\varepsilon[h/X] \qquad \dfrac{\dfrac{\dfrac{\lambda y.\lambda x.hug(x,y) : A \multimap B \multimap C \qquad [z : A]^1}{\lambda x.hug(x, z) : B \multimap C}\multimap\varepsilon,\Rightarrow_\beta \qquad kim : B}{hug(kim, z) : C}\multimap\varepsilon,\Rightarrow_\beta}{\lambda z.hug(kim, z) : A \multimap C}\multimap_{\mathcal{I},1}}{\forall y.[person(y) \rightarrow hug(kim, y)] : C}\multimap\varepsilon,\Rightarrow_\beta$$

Note that $'\forall X'$ in the denotation for "every" means that *any* of a given resource may be consumed, rather than all resources at once ((Crouch and van Genabith 2000:89)): it tells us that some property holds of anything in the domain and allows selection and utilisation of that resource as need be. This chimes with the resource-based nature of linear logic, since we don't wish to consume all our resources in one go, using up all resources. The universal elmination rule is as follows:

Universal Elimination

$$\dfrac{\begin{array}{c}\vdots\\ \forall X.A\end{array}}{A[h/X}\,\forall_{\mathcal{E}}[h/X]$$

2.2.3 Example III: Scope Ambiguity

(2.5) Everybody hugged somebody.

Expression	Type	Meaning Constructor
hugged	$e \to (e \to t)$	$\lambda y.\lambda x.hug(x,y) : (A \multimap (B \multimap C)$
everybody	$(e \to t) \to t$	$\lambda P.\forall y.[person(y) \to P(y)] : (B \multimap C) \multimap C$
somebody	$(e \to t) \to t$	$\lambda P.\exists x.[person(x) \wedge P(x)] : (A \multimap C) \multimap C$

Simplified $\exists > \forall$ proof for 'Everybody hugs somebody':

$$\dfrac{\lambda P.\exists x.[person(x) \wedge P(x)] : (A \multimap C) \multimap C \qquad \dfrac{\dfrac{\lambda P.\forall y.[person(y) \to P(y)] : (B \multimap C) \multimap C \qquad \dfrac{\dfrac{\lambda y.\lambda x.hug(x,y) : (A \multimap (B \multimap C) \qquad [z:A]^1}{\lambda x.hug(x,z) : B \multimap C} \multimap_{\mathcal{E}}, \Rightarrow_\beta}{\forall y.[person(y) \to hug(y,z)] : C} \multimap_{\mathcal{E}},}{\lambda z.\forall y.[person(y) \to hug(y,z)] : A \multimap C} \multimap_{\mathcal{I},1}}{\exists x.[person(x) \wedge \forall y.[person(y) \to hug(y,x)]] : C} \multimap_{\mathcal{E}}, \Rightarrow_\beta$$

Simplified $\forall > \exists$ proof for 'Everybody hugs somebody':

$$\dfrac{\lambda P.\forall y.[person(y) \to P(y)] : (B \multimap C) \multimap C \qquad \dfrac{\dfrac{\lambda P.\exists x.[person(x) \wedge P(x)] : (A \multimap C) \multimap C \qquad \dfrac{\dfrac{\dfrac{\lambda y \lambda x.hug(x,y) : A \multimap (B \multimap C) \qquad [y_1 : A]^1}{\lambda x.hug(x,y_1) : (B \multimap C)} \multimap_{\mathcal{E}}, \Rightarrow_\beta \qquad [z:B]^2}{hug(z,y_1) : C} \multimap_{\mathcal{E}}, \Rightarrow_\beta}{\lambda y_1.hug(z,y_1) : (A \multimap C)} \multimap_{\mathcal{I},1}}{\exists x.[person(x) \wedge hug(z,x)] : C} \multimap_{\mathcal{E}}, \Rightarrow_\beta}{\lambda z.\exists x.[person(x) \wedge hug(z,x)] : (B \multimap C)} \multimap_{\mathcal{I},2}}{\forall y.[person(y) \to \exists x.[person(x) \wedge hug(y,x)]] : t} \multimap_{\mathcal{E}}, \Rightarrow_\beta$$

In the glue proofs above denotations are not assigned to each constituent of the syntax tree. For example, in 2.2.3 no part of the glue proof seems to correspond to the verb phrase "hugs somebody" and it is not clear what part of the syntax tree the second and third line of the glue proof correspond to. In fact, glue semanticists want semantics to operate with a greater degree of freedom from syntax. (for a version of glue semantics for binary branching syntax trees which accepts this degree of freedom, see Gotham (2015).)

Nevertheless, Gl-TAG establishes a much tighter correspondence between the syntax and the semantics and satisfies the "Principle of Interpretability" (Heim and Kratzer (1998) according to which all nodes in a phrase structure tree must be in the domain of the interpretation function. In Gl-TAG, a function takes a node of the tree to part of a glue proof, which receives its model-theoretic interpretation. In this way we adopt a modified principle of interpretability according to which all nodes in a phrase structure tree are in the domain of a function whose value is some particular set of nodes of a (semantic glue) tree. A semantic tree is simply the parse tree of a glue proof. This function is, moreover, stable: verb phrases in the syntax trees are always mapped to the same region of semantic trees, and quantifier phrases are similarly mapped. The function mapping a mother node to a part of the semantic tree is a function that applies to some region of the semantic tree to which one of daughters corresponds and some region of the semantic tree to which the other daughter corresponds, to yield a semantic tree that incorporates both of these tree regions. Since there is then a function from parts of these semantic trees to their model theoretic interpretation, our semantics systematically maps parts of the syntactic structure to their model theoretic interpretation. The semantic trees act as an intermediary language, much as IL was for Montague an intermediary language. Whether the resulting semantics is compositional is a question we shall pursue in the next chapter. We now give some examples of semantic trees, and then specify the operations of TAG that apply to them.

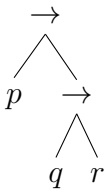
2.3 Glue parse trees and TAG

2.3.1 Parse trees

In logic, parse trees (Chiswell and Hodges (2007)) represent the derivational history of formulas. The following parse trees describe the formation of $p \to q$ and $p \to (q \to r)$:

(3) a.

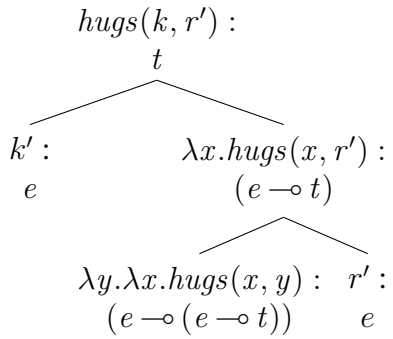
b.



Since each step of a glue proof in implicative linear logic proceeds by either functional application or by abstraction, one daughter of a node is labeled by a function and the other by its argument, and thus every glue proof can be represented as a binary tree. (In the case of abstraction, the CHI guarantees (see 2.2) that there is a function combining the reintroduced hypothesis and the denotation of its sister.)

Consider a simple sentence, "Kim hugs Robin". This has the following binary tree, where *e* and *t* are linear logic formulas corresponding to types *e* and *t* (from now on we use e and t as variables for linear logic formulae corresponding to types):

(4)



We establish a mapping from any given node of the syntax tree to its denotation, as follows: first we assign a part of a semantic tree as denotation of a given terminal node in the syntax tree; then we specify (i) rules for combining the semantic trees (these correspond to rules that combine constituents in the syntax tree) and (ii) rules that govern how non-branching nodes in the syntax trees inherit the semantic values they directly dominate. The following rules govern our syntactic trees:

Composition Rule 1a:

Functional Application (FA)

Let γ be a tree whose only two subtrees are α and β. Then if $\alpha \in$ *dom* ($[\![\ ]\!]^{\mathcal{M},\,g}$) and $\beta \in$ *dom* ($[\![\ ]\!]^{\mathcal{M},\,g}$) and β is a function such that $[\![\alpha]\!]^{\mathcal{M},\,g} \in$ *dom* ($[\![\beta]\!]^{\mathcal{M},\,g}$), then $\gamma \in$ *dom* ($[\![\ ]\!]^{\mathcal{M},\,g}$) and

$[\![\gamma]\!] = [\![\beta]\!]([\![\alpha]\!])$.

Note that the order of α and β in the tree doesn't matter (as (i) and (ii) below show):

(i) $\left[\!\!\left[\begin{array}{c} \gamma \\ \alpha \quad \beta \end{array} \right]\!\!\right]^{\mathcal{M},\,g} = \left[\!\!\left[\beta \right]\!\!\right]^{g} ([\![\alpha]\!]^{g})$ (ii) $\left[\!\!\left[\begin{array}{c} \gamma \\ \beta \quad \alpha \end{array} \right]\!\!\right]^{\mathcal{M},\,g} = \left[\!\!\left[\beta \right]\!\!\right]^{g} ([\![\alpha]\!]^{g})$

Composition Rule 2:
Non-Branching Nodes (NN)

If α is a tree whose only subtree is β, then $[\![\alpha]\!] = [\![\beta]\!]$.

$$\left[\!\!\left[\begin{array}{c} \alpha \\ | \\ \beta \end{array} \right]\!\!\right]^g = [\![\beta]\!]$$

Composition Rule 3:
Lexical Terminals (LT)

If α is a terminal node, then its translation is specified in its lexical entry:

$$\begin{array}{c} \ldots \\ | \\ \alpha \end{array} = [\![\alpha]\!]$$

Such *type driven* rules (Klein and Sag (1985)) require the nodes of a semantic tree to combine in a certain way. But, unlike Heim and Kratzer (1998), who stipulate these additional rules, glue proofs require them to hold automatically: with the exception of composition rule 2, they actually flow from the fragment of linear logic used, since each step of a proof proceeds by functional application or by abstraction and one of the binary branches always corresponds to a function and the other to its argument. Consequently, we can deduce the semantic value of a given mother node from its daughters, and we get the type-driven rules above for free.

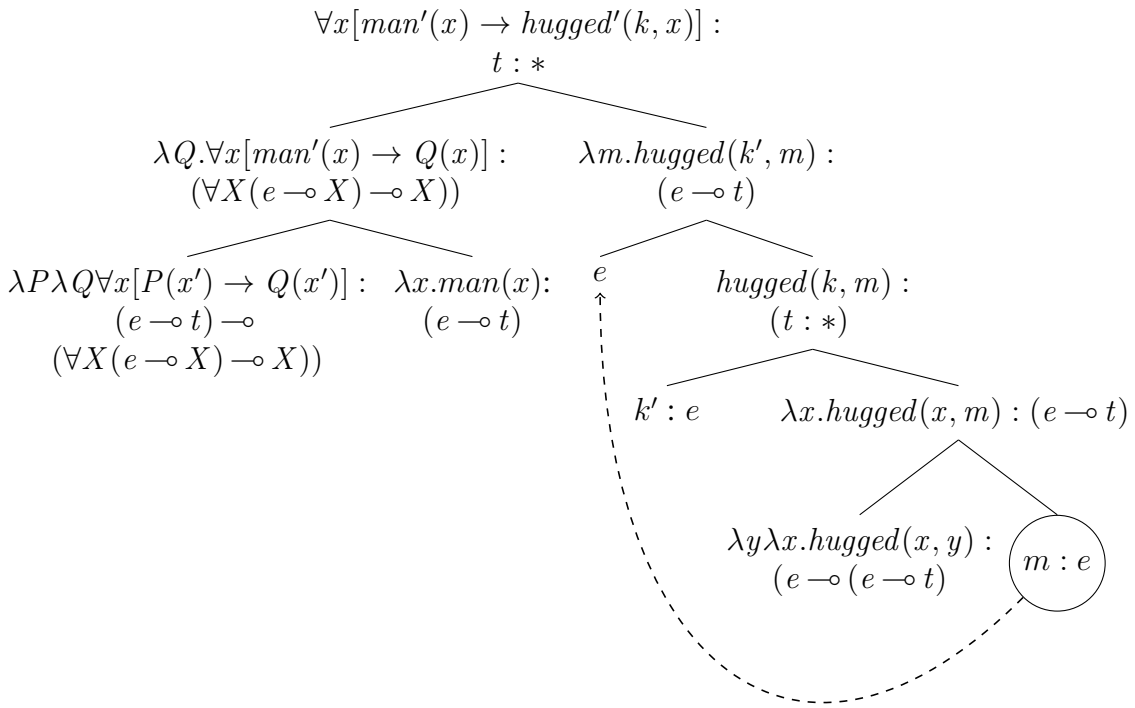
To give a taste of what a more complicated example would look like, consider *Kim hugged every man*, whose parse tree and the original proof are here presented:

(5)

Proof for (2.4), abridged for $\Rightarrow_\beta$:

$$\dfrac{\dfrac{\lambda P.\forall y.[man(y) \rightarrow P(y)] : \forall X.(e \multimap X) \multimap X}{\lambda P.\forall y.[man(y) \rightarrow P(y)] : (e \multimap t) \multimap t}\,\forall\mathcal{E}[h/X] \qquad \dfrac{\dfrac{\dfrac{\lambda y.\lambda x.hug(x,y) : e \multimap (e \multimap t) \qquad [z:e]^1}{\lambda x.hug(x,z) : e \multimap t}\,\multimap\mathcal{E},\Rightarrow_\beta \qquad kim : e}{hug(kim,z) : t}\,\multimap\mathcal{E},\Rightarrow_\beta}{\lambda z.hug(kim,z) : e \multimap t}\,\multimap\mathcal{I},1}{\forall y.[man(y) \rightarrow hug(kim,y)] : t}\,\multimap\mathcal{E},\Rightarrow_\beta$$

(6)



We discuss the details of such trees later, but for now, note that we have enclosed the hypotheses in circles, and drawn arrows which point to the part of the tree structure at which the abstraction operation is carried out (the implication introduction operation in the linear logic formula). The fundamental idea behind Gl-TAG is to map constituents of the syntax tree to parse trees of the above form. In the same way that constituents group together to form larger constituents, parse trees group together to form larger constituents. Corresponding to these ways in which parts of parse trees combine are semantic functions that combine the denotations of nodes of the parse trees.

Before describing a simple TAG grammar, and introducing our semantics, we will address the question of how the parse trees of glue proofs are obtained from the (surface) syntactic structure of the sentence. Heim and Kratzer (1998) assume LFs are generated by the syntax. But in Gl-TAG this is not the case. What is the relevant relationship? Barker (Barker and Jacobson (2007)) describes the relationship well (albeit he is speaking of continuations):

> Clearly, continuations exist independently of any framework or specific analysis, and all occurrences of expressions have continuations in any language that has a semantics...*they are present whether we attend to them or not*. The question...then, is...how natural language expressions do or don't interact with them. (Barker, 2002: 5).

We take the following position. Given an assignment of lambda terms to expressions in a sentence, a glue proof is generated. The proof is a mathematical object that we "attend to or not", as are

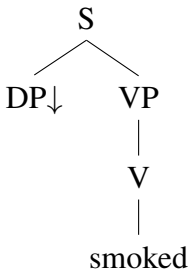
the parse trees of glue proofs. So the question of how they are derived is transparent: they must exist, given the lexical assignments we have given, and we can explore how they relate to natural language sentences. This is a rather desirable situation; after all, the principles of LF construction are not so robust as to be mathematically necessary. We therefore need not have substantial doubts about the derivability of a glue tree from lexical assignments made to expressions in a sentence, whereas the syntactic processes that generate LFs are mired in controversy.

2.3.2 TAG grammars

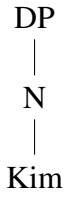
TAG grammar (Joshi and Schabes (1997)) permits us to construct syntax trees by means of various operations which piece trees together. A TAG grammar has a set of initial trees which are incomplete, in that they may contain empty terminal nodes. The terminal nodes marked with $(\downarrow)$ indicate we may substitute in other trees whose root share the categorial label of the terminal node.

Here is an elementary example. Suppose we have the following initial trees:

(7) a.

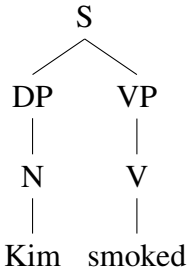


b.



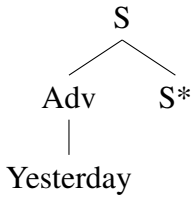
The initial tree, (7a), contains an empty terminal node, with a $(\downarrow)$ symbol. This indicates that we may substitute a tree with a root node DP into this position and so we can produce the following tree (substituting the tree in (7b)):

(8)

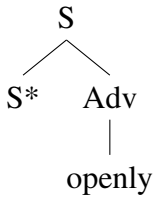


Another operation, adjunction, expands initial trees by means of a special class of auxiliary trees. An auxiliary tree is a tree whose root is labeled identically to some node along its frontier, the foot node (which is marked with an asterisk). Take the following auxiliary trees:

(9) a.

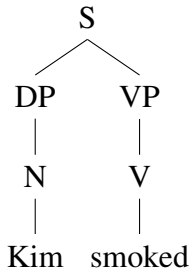


b.

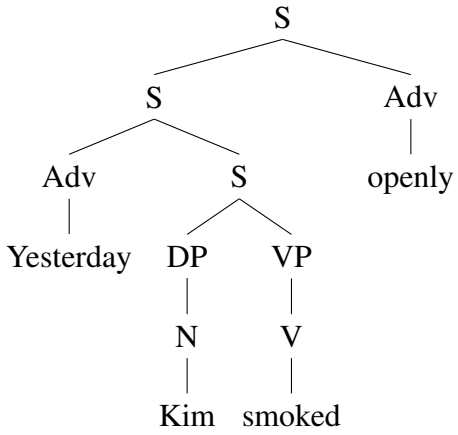


By adjunction, an auxiliary tree breaks up and inserts itself into an initial tree one of whose nodes is labelled by the same label (in this case, S) that labels the root and foot of the auxiliary tree. Two applications of adjunction derive (10b) from (10a):

(10) a.



b.



The class of Tree-adjoining grammars, and the operations of substitution and adjunction are now defined.

Definition 18 (see Joshi and Schabes (1997) for a similar definition):

A tree-adjoining grammar (TAG) consists of a quadruple $(\Sigma; I; A; t)$, where

- Σ is a finite set of symbols for types such that a member of Σ is either of type e, t or a type derived from e and t via the rules of type formation (see the technical prelimary)
- t is the starter symbol, and a subset of Σ
- I is a set of finite trees, *initial trees*, such that:

  - interior nodes are labeled by a member of Σ; (The initial trees we consider are simply terminal frontier nodes, which are vacuously interior nodes)

  - the nodes on the frontier of initial trees are labeled by a member of Σ; certain terminal symbols on the frontier of the trees in I are marked for substitution by a down arrow $(\downarrow)$;

- A is a finite set of finite trees, called auxiliary trees, characterized as follows:

  - interior nodes and nodes on the frontier are labeled by a member of Σ;

- Certain symbols on the frontier of the trees in A are marked for substitution except for one node, called the foot node, which is marked with an asterisk (*); the label of the foot node must be identical to the label of the root node.

Definition 19 Adjoining builds a new tree from an auxiliary tree β and a tree α (α is any tree, initial, auxiliary or derived). Let α be a tree containing a non substitution node n labeled by X and let β be an auxiliary tree whose root node is also labeled by X. The resulting tree, γ, obtained by adjoining β to α at node n is built as follows:

1. the sub-tree of α, dominated by n, call it t, is excised, leaving a copy of n behind.
2. the auxiliary tree β is attached at the copy of n and its root node is identified with the copy of n.
3. the sub-tree t is attached to the foot node of β and the root node of t (i.e. n) is identified with the foot node of β.

Definition 20 Substitution takes place on nodes of the frontier of a tree. When substitution occurs on a node n, the node is replaced by the tree to be substituted. When a node is marked for substitution, only trees derived from initial trees can be substituted for it.

Gl-TAG utilises the TAG operations of adjunction and substitution defined above. In particular, quantifier phrases in glue proofs correspond to auxiliary trees in parse trees and the scope of a quantifier in the glue proof corresponds to the position at which an auxiliary tree structure has adjoined in the parse tree.

But there are two complications. Firstly, our analysis actually uses tree-local MC-TAG (see Weir (1988), for its formal properties). An MC-TAG has a finite set S of sets $s_1, ..., s_n$ of multi-component trees. Multi-component trees are sets containing a single tree s decomposed into parts. Any operation applying to s must adjoin or substitute its parts simultaneously to an elementary tree, and we can define functions that are only defined if these operations are simultaneous.

MC-TAG has the same weak and strong generative capacity as TAG and has been used for identical purposes for which we use it here; namely, to allow for quantifier scope and discontinuous syntactic dependencies (see (Han and Hedberg 2008:356)).

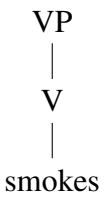
Gl-TAG depends on a specific function, the SNAP function. Suppose we substitute an argument into a node n of a tree. The range of this operation is in the domain of the function γ which is its sister in the tree. The range of γ is the domain of the mother node of n. Since this holds all the way up a tree, we can in fact compose all these functions into one huge function, the SNAP function. Thus when we substitute, we also apply function application up the tree as far as we can go. Furthermore, when a quantifier tree adjoins to a position, it may adjoin again. We specify that the quantifier first adjoins to the nearest t node. But since, if it adjoins to another place after this moment, the range of the initial adjunction α_0 operation is the the domain of the next operation α_1 of adjunction, and the range of α_1 is the domain of α_2, we can compose these operations into a single function. We then require that substitution is only defined when the ultimate adjunction in the series occurs (for if not, the substition would have to be supplied once for each adjunction). Gl-TAG thus uses TAG functions as semantic operations.

2.4 Gl-TAG

2.4.1 Simple in/transitive sentences

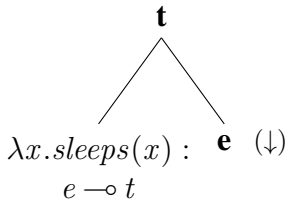
In Gl-TAG, the semantic value of a mother node in the syntax tree is a function of the semantic values of its daughters. But the function that combines the semantic values of the two daughters is a function that combines two glue proof parse trees, one corresponding to the DP and one corresponding to the VP (this function from syntax to semantics trees is denoted $\lceil\ \rceil$). Suppose we have the syntax tree for "smokes":

(11)



Intransitive verbs are mapped to TAG initial trees of the following kind:

(12)



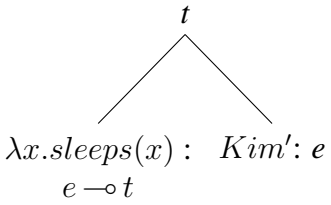
e and **t** are in bold to mark the fact that they are are variables over linear logic formulas that correspond to simply typed λ calculus formulas of type *e* and *t* (the down arrow next to the **e** indicates a substitution site). Suppose we substitute in an appropriate meaning constructor, such as the following lexical entry:[2]

(13) Kim': e

Since the node is labelled **e**, this is a legitimate substitution, provided that we suppose that substitution is only sensitive to the formula of glue logic on the right hand side of the meaning constructor. Substitution of Kim' for δ produces the following tree:

[2]Our procedure is far from ad hoc, and is in fact required by the operation of substitution in TAG. In TAG, substitution requires that the categorial label into which a substitution is being made is a variable over syntactic objects of that category. Hence, our insisting that this is the case is in fact explained by the nature of the substitution operation we are applying. Furthermore, our distinguishing the nodes substituted into the tree from the initial tree itself receives support from consideration of the underlying linear logic. In the underlying liner logic formula the hypotheses are distinguished from formulae to which they apply. Thus our requirement that **e** and **t** in the parse tree being variables over objects of type e and t is well supported by the theoretical tools we are using.

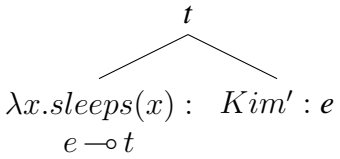
(14)



Since the range of the substitition function is the domain of the function $\lambda x.sleeps(x)$, we form one SNAP function, the composition of the substitution operation with function application:

(15)

$\lambda x.sleeps(x)(Kim')$ (= Kim sleeps) :



Each word in a sentence has a lexical entry, its *core semantic value* (for $sleep$ this is $\lambda x.sleeps(x)$). However, in Gl-TAG, an expression e has a *peripheral semantic value* (the **e** and **t** nodes in (12)). The peripheral semantic value of e is the set of all the rest of the node labels on the tree (without the core semantic value). So in the case of "sleep", the peripheral semantic value of "sleep" is $\{\mathbf{e}, \mathbf{t}\}$, where **e** and **t** are variables over linear logic formulae that correspond to simply typed λ calculus formulae of type *e* and *t* respectively.

The semantic value of an expression is the set containing its core semantic value, a lexical entry, and its peripheral semantic value, and the entire tree is an elementary tree. [3] The present view of the semantic value of $sleep$ brings out nicely the fact that the core semantic value of $sleep$ will enter into inference with the semantic value for another term to produce the semantic value of a sentence. In fact, since the elementary tree exhibits the dependency of the verb as a predicate on its two arguments, it satisfies the semantic analogue, as regards predicate structure, of what Frank (2004) has termed "The fundamental hypothesis of TAG-based work...that dependencies must be expressed within the structural context of an elementary tree". Our proposal that verbs are elementary trees of the sort described therefore fits nicely with the TAG-theoretic conception of elementary trees.

A number of analogies with alternative theories clarify our distinction between core and peripheral semantic value. For example, the distinction gives rise to a conception of semantic value similar to that of a continuation. According to (Kelsey et al. 1998:p.71) "a continuation represents the entire (default) future for the computation." In Gl-TAG the semantic tree for a verb represents

[3] Instead of distinguishing core and peripheral semantic value, we could simply equate the semantic value of a verb with the (model theoretic interpretation of) a set of meaning constructors. The problem with this approach is that it fails to explain where the premises of the glue proof are coming from. The intuitive answer to this question is that the premises are lexical entries for words in the sentence, including verbs. But this answer is not available if we simply identify the semantic value of a verb with a set of meaning constructors. For this reason the distinction between core and peripheral semantic value is conceptually superior.

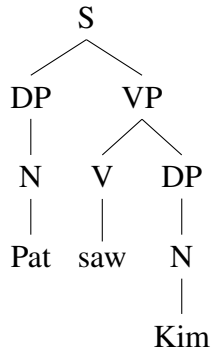
the future of the semantic computation when it is incomplete. However, unlike in (Barker (2002)), the semantic value of an expression is not identified with a continuation. Thus, we avoid equating semantic values with continuations, but we can maintain the simple lexical entries of Heim and Kratzer (1998) for core semantic values, whilst representing aspects of the continuation approach.

A second way of understanding the contrast between core and peripheral semantic value is as analogous to the distinction between heads and non heads in X-bar theory. In X-bar theory a head projects a maximal projection. The non-head nodes of the maximal projection are all members of this projection. The lexical entry in the semantic tree for an intransitive is the semantic analogue of the head of a maximal projection and the peripheral semantic value as being the semantic analogue of the maximal projection of a head.

There are however disanalogies; the lexical entry for $sleep$ is an independent item in the grammar, whereas in X-bar theory heads do not exist without their projections (unless we allow a lexical entry to have itself as a trivial projection).

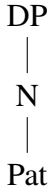
We now consider transitive clauses of the following kind:

(16)

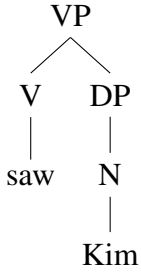


We will describe a TAG function that maps a semantic tree for the following syntactic tree of a VP ((17b)) and a semantic tree for the following subject DP ((17a)) to a semantic tree for the syntactic tree in (16):

(17) a.



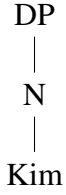
b.



We must first map the semantic tree for the object DP ((18b)) to the semantic tree for the V ((18a)):

(18) a.

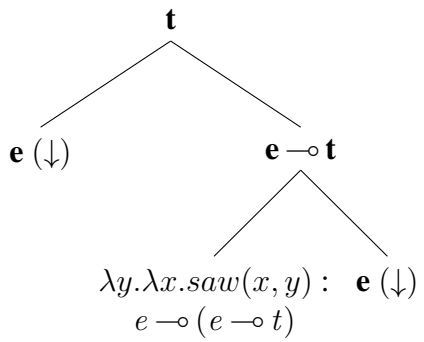
b.



(18a)

is mapped to the following parse tree:

(19)



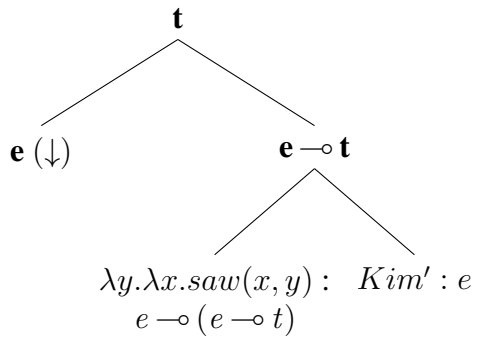
(18b)

is mapped to the following trivial semantic tree, a lexical entry:

(20) $$Kim' : e$$

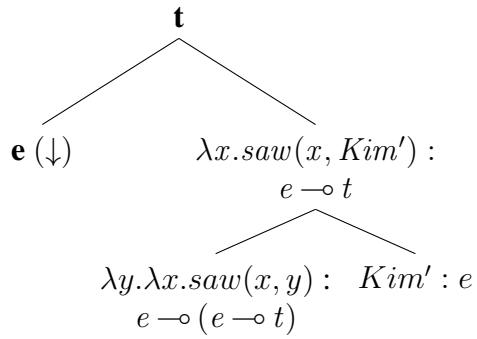
So the VP is mapped to the following semantic tree, by substituting (20) into the rightmost branch of the semantic tree:

(21)



Since the range of the substitition function is the domain of the function $\lambda y.\lambda x.saw(x, y)$, we form one SNAP function, the composition of the substitution operation with function application:

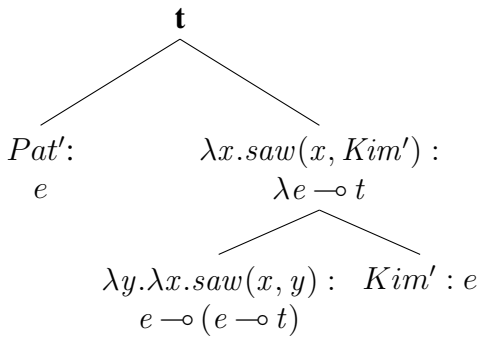
(22)



Note that we cannot first substitute a lexical entry into the leftmost node in semantic tree for the verb, since the function node **e** ⊸ **t** needs to assume a specific value before it can combine with its sister, which it can't until the bottom-most variable in the semantic tree is substituted with an appropriate meaning constructor.

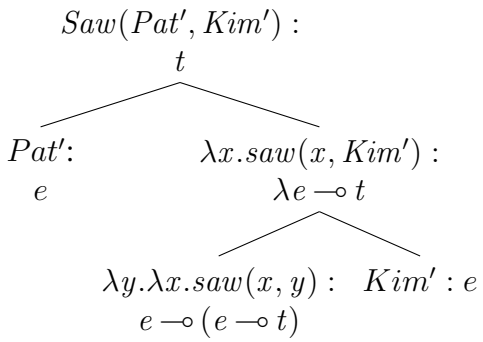
We next combine the semantic tree for the DP subject with the semantic tree for the VP. This is done by substituting the semantic tree for Pat into the leftmost branch of the semantic tree for the VP, producing the following semantic tree:

(23)



Since the range of the substitition function is the domain of the function $\lambda x.saw(x, Him')$, we form one SNAP function, the composition of the substitution operation with function application:

(24)



We now consider quantifiers.

2.4.2 Quantifiers

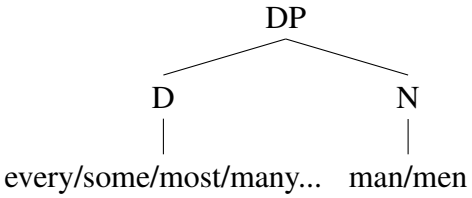
A theory of quantification ought to answer the following questions (see Von Stechow (2009), Barker (2002)):

1. How is the problem of the object resolved? (see chapter 1, section 1.3)
2. *Scope ambiguity*: How is Qscope ambiguity accommodated?
3. How is variable binding achieved?
4. *Duality of NP meaning*: What (if anything) unifies the meanings of quantificational vs. non-quantificational NPs?
5. *Scope displacement*: Why does the semantic scope of a quantified NP sometimes differ from its syntactic scope?[4]

During the course of the following sections we will see how each of these questions are given a distinctive answer in our framework.

We now want to map the following type of syntactic tree to a semantic tree:

(25)



First we need to show what semantic trees the following parts of the syntax correspond to:

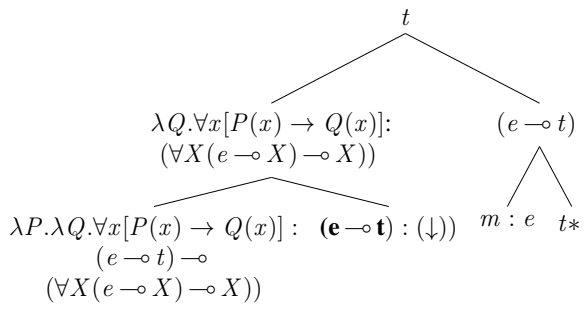
(26) a.

b.

We then show how they combine. Following (Barker and Shan (2014), Bernardi (2010), Joshi et al. (2007)) in Gl-TAG a quantifier phrase has two parts; a part that satisfies a given predicate and a part that accounts for scopal properties. For this reason, the semantic value of a quantifier phrase is in fact an elementary tree set of the following form (in the sense described above (p41), consisting of a lexical entry and an auxiliary tree (one of whose foot nodes is labelled with the lexical entry), whose members are simultaneously adjoined and substituted respectively to the verb:

[4]Barker distinguishes 2 from 5 for two reasons: (i) on some theories (May 1985) relative scope is determined by a separate mechanism from scope displacement and (ii) the existence of scope ambiguity has been disputed but the existence of scope displacement has not. Ultimately we will show that our theory, like Barker's, can give a unified answer to 2, 4 and 5. Whether this is an advantage or not depends on whether a unified explanation of the three phenomena is empirically warranted, a question we do not have space to consider.

$$\left\{ \begin{array}{l} m : e \end{array} \right.$$



$$\left. \begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array} \right\}$$

(2.6)

The core semantic value of "every" is $\lambda P.\lambda Q.\forall x[P(x) \multimap Q(x)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$. The peripheral semantic value of "every" is the set containing the meaning constructors of the other nodes in the elementary tree set.

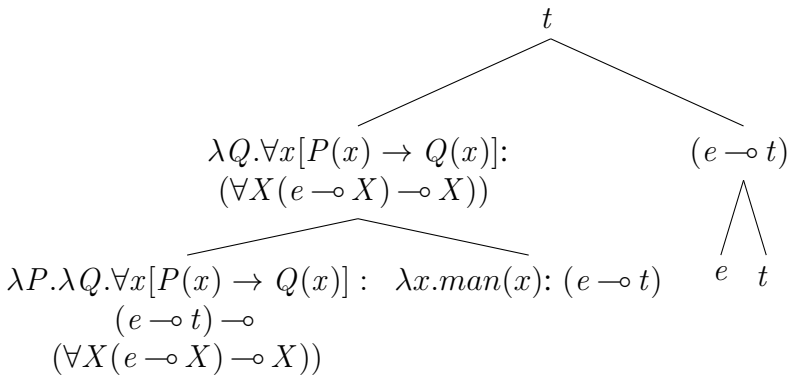
A quantifier is an auxiliary tree (since its root is the same as one of the footnodes marked with an asterisk) and can therefore adjoin to the t nodes of other trees, a fact of crucial importance.

Nominal predicates like man have lexical entries of the following kind:

(27) $\lambda x.man(x)$: $(e \multimap t)$

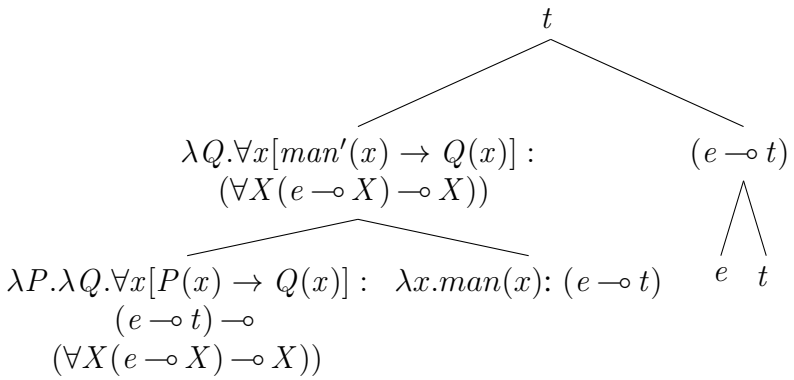
The semantic tree for "Every man" is derived by substituting the semantic tree for "man" into the semantic tree for "Every", at the node adjacent to the core semantic value of "every" in (59). This derives the following semantic tree:

(28)



Since the range of the substitition function is the domain of the function $\lambda P.\lambda Q.\forall x[P(x) \rightarrow Q(x)]$, we form one SNAP function, the composition of the substitution operation with function application:

(29)



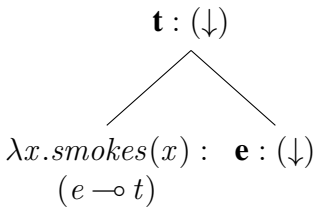
One nice feature of this analysis is that nominal predicates such as "man" are treated similarly to how we treated names above, albeit they have a distinct type. This is the reflex in the semantic trees, of the fact that "man" and "John" are both noun phrases.

Paul Elbourne (p.c) asked how we prevent "sleep" from being substituted in where $\lambda x.man(x)$ has been substituted, given they both have the same type. There are two responses for this. Firstly, "Every sleep" is not a syntactic structure that can be interpreted. Since Gl-TAG takes well-formed syntactic structures as its input, this issue doesn't arise, since it only for every well-formed operation of the syntax we provide an operation in the semantics.

For selectional restrictions of the non-syntactic kind, we can always adopt substitution constraints, familiar from TAG theory, which impose selectional restrictions on what can be substituted and where. In the case of (28), for example, if we had evidence that "Every sleep" was ruled out semantically by the selectional requirements of "Every", we could say that an intrasitive verb cannot be substituted in where $\lambda x.man(x)$ has been substituted, since this would violate the selectional restrictions that $every$ imposes on its complement; namely that it be nominal. [5]

How does the elementary tree set for "Every man" combine with an intransitive VP such as $smoke$, the following tree?

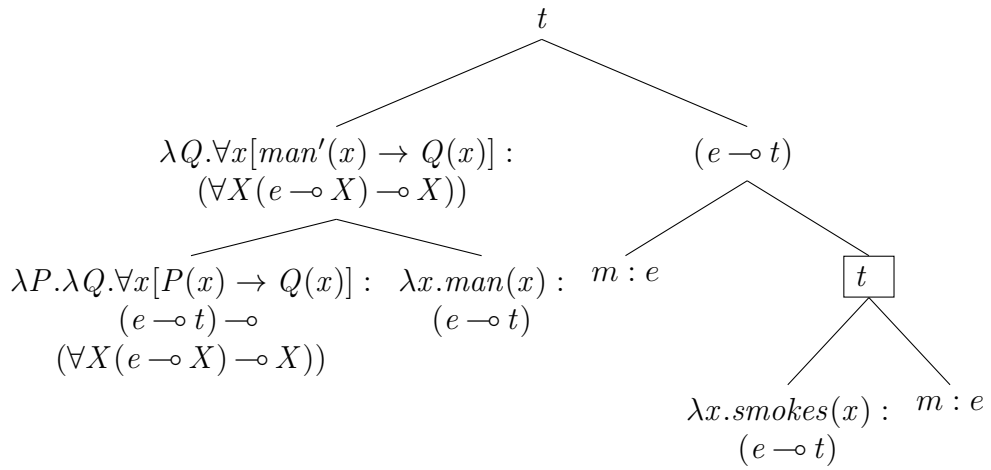
(30)



Since the semantic tree for "Every man" has a foot node and root note labelled t, we adjoin it to the root node of $smokes$. But since we are taking *every man* to have two parts (which are the members of the same elementary tree set) we require one operation which both adjoins the auxiliary tree and substitutes the hypothesis that corresponds to it into the node of the tree, for the verb. The existence of such an operation is guaranteed by the nature of multi-component sets in MC-TAG, and results in the following structure:

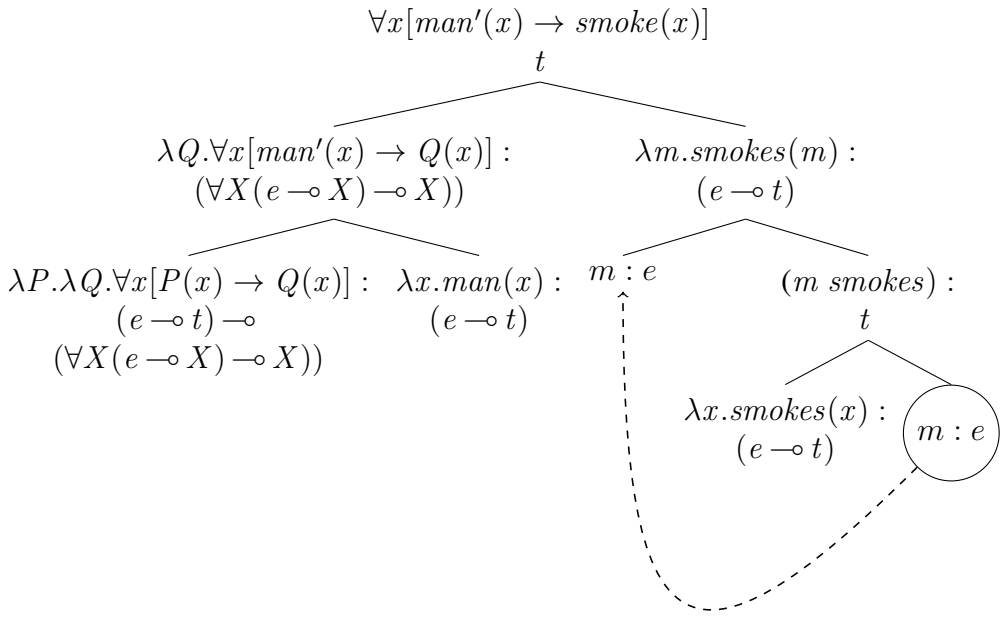
[5] We leave the formulation of such constraints for further research.

(31)



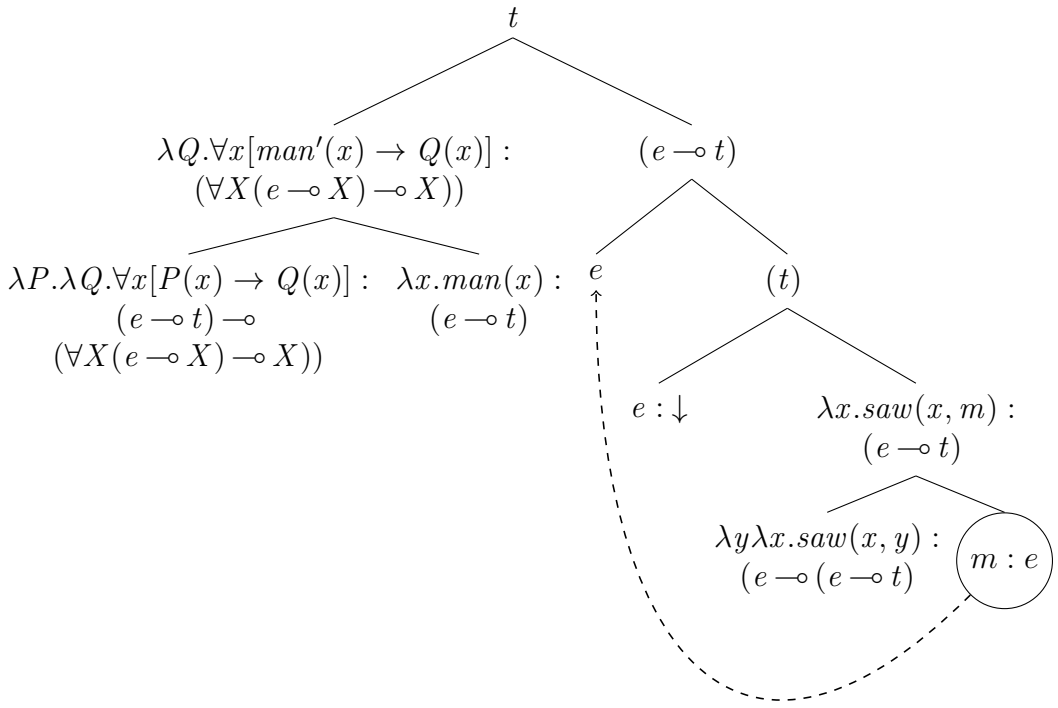
Since the range of the substitution operation is the domain of $\lambda x.smokes(x)$ and the range of $\lambda x.smokes(x)$ is the domain of the t node, and the range of the abstraction function is the domain of $\lambda Q.\forall x[man'(x) \rightarrow Q(x)]$, we can form the composition of all these operations. Thus for every adjunction/substitution operation in our semantics there exists a SNAP function, the composition of the substitution operation with a chain of functions (and thus we have a joint adjunction/SNAP operation) that results in the semantic value of the tree:

(32)



We now derive the semantic value of "Kim saw every man". "Saw every man" corresponds to the following semantic tree:

(33)



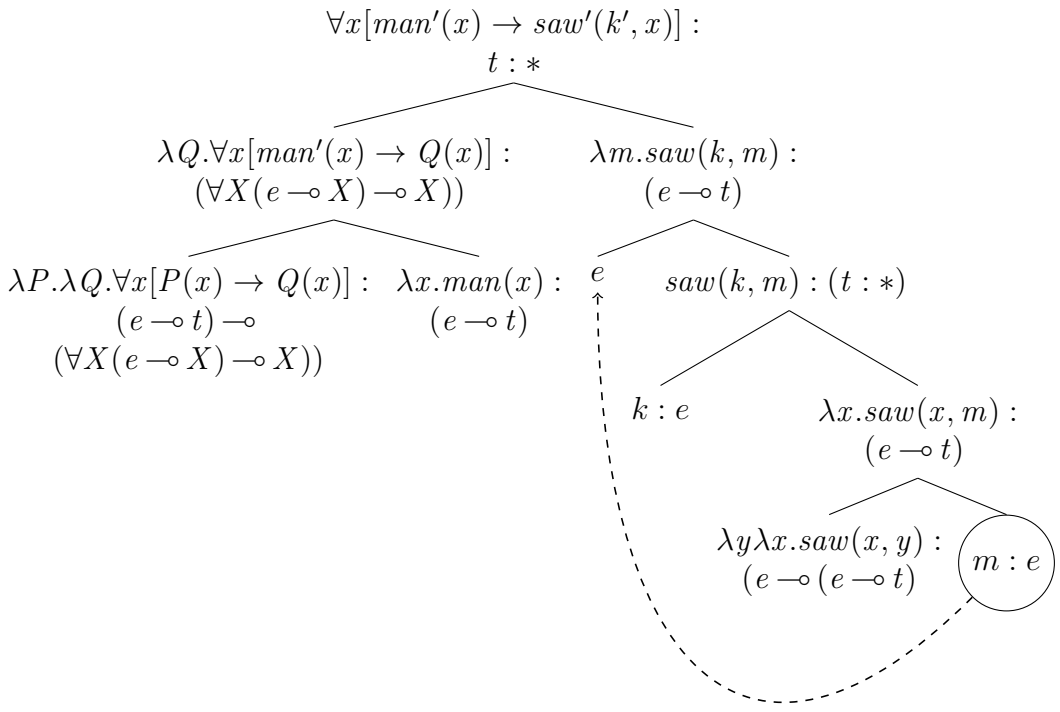
The tree is built by an operation on the elementary tree set for a quantifier, which simultaneously adjoins one member of this set, the semantic tree for "every man", to the semantic tree for "saw" and substitutes another member of this set, m, in the tree for saw. As before, the substitution operation can be composed with the function $\lambda y.\lambda x.saw(x, y)$. Note that in Gl-TAG a VP does not have a core semantic value in an incomplete tree. Furthermore, the semantic value of a VP is not obtained by applying the core semantic value of the verb to the core semantic value of the object. Nevertheless, the semantic value of the VP is a function of the semantic value of the direct object.

The tree illustrates our solution to the problem of the object (see chapter 1, section 1.3). The type clash between the core semantic value of a transitive verb and a quantifier is resolved once we enrich our conception of meaning to accommodate the core and peripheral semantic value of a verb and a quantifier, and utilise the machinery of MC-TAG by putting the parts of a quantifier into an elementary tree set. When considering both the core and peripheral semantic value of a transitive verb and a quantifier, the problem of the object disappears. Thus our solution to the problem of the object flows naturally from the conception of semantic value we have articulated.

The tree also illustrates our answer to the question of scope displacement. Scope displacement follows from the dual nature of quantifiers, whose semantic values comprise a part which satisfies a predicative part and a scopal part. This ensures that the scopal part of a quantifier in the syntax tree does not match the position of the quantifier in the syntax. Again, without viewing the semantic value of a quantifier in this dual way, such a solution would not be possible, thus our conception of semantic value allows a simple answer to the question of displacement. Thus our solution to the problem of the object flows naturally from the conception of semantic value we have articulated.

We now substitute "Kim" into this structure:

(34)



Note that the core semantic value of the VP can be obtained from the completed tree by substituting a variable (σ) over expressions for k' in the root node denotation. (We have also explored an equivalent system to Gl-TAG in which the core semantic value is reflected also in incomplete trees, but this needs development).

The only special rule of semantic combination operant in our fragment (apart from the general rules governing trees we discussed above) is thus the following:

(35)

Composition Rule for our fragment
Parse tree combination

Let γ be a syntax tree whose only two daughters are α and β and suppose that $\lceil\rceil^{\mathcal{M}}$ is the SNAP function from syntax trees to semantic trees. Then if $\alpha \in$ *dom*($\lceil\rceil^{\mathcal{M},\,g}$) and $\beta \in$ *dom*($\lceil\rceil^{\mathcal{M},\,g}$), then:
(i) If α is a name or a nominal predicate, substitute $\lceil\alpha\rceil$ in $\lceil\beta\rceil$ (or If β is a name or a nominal predicate, substitute $\lceil\beta\rceil$ in $\lceil\alpha\rceil$), and compose the substitution operation with the function f_1 on the sister node of the predicate, and compose the value of f_1 with its sister (repeating this process as far up the tree as possible and so forming a single SNAP function)
OR
(ii) adjoin $\lceil\alpha\rceil$ to a t node in $\lceil\beta\rceil$ (or adjoin $\lceil\beta\rceil$ to a t node in $\lceil\alpha\rceil$) and substitute the hypothesis in its tree set into the subject (object) position of the verbal semantic tree. (Again, composing the substitution operation with the function f_1 on the sister node of the predicate, and composing the value of f_1 with its sister (repeating this process as far up the tree as possible and so forming a single SNAP function)).

From now on we treat names and quantifiers as equivalent, since they are interderivable in the underlying linear logic. This has the advantage that we can treat both names and quantifiers as adjoining structures governed by condition (ii) above, thus simplifying our semantic theory, and we can shift quantifiers to names as and when is necessary.

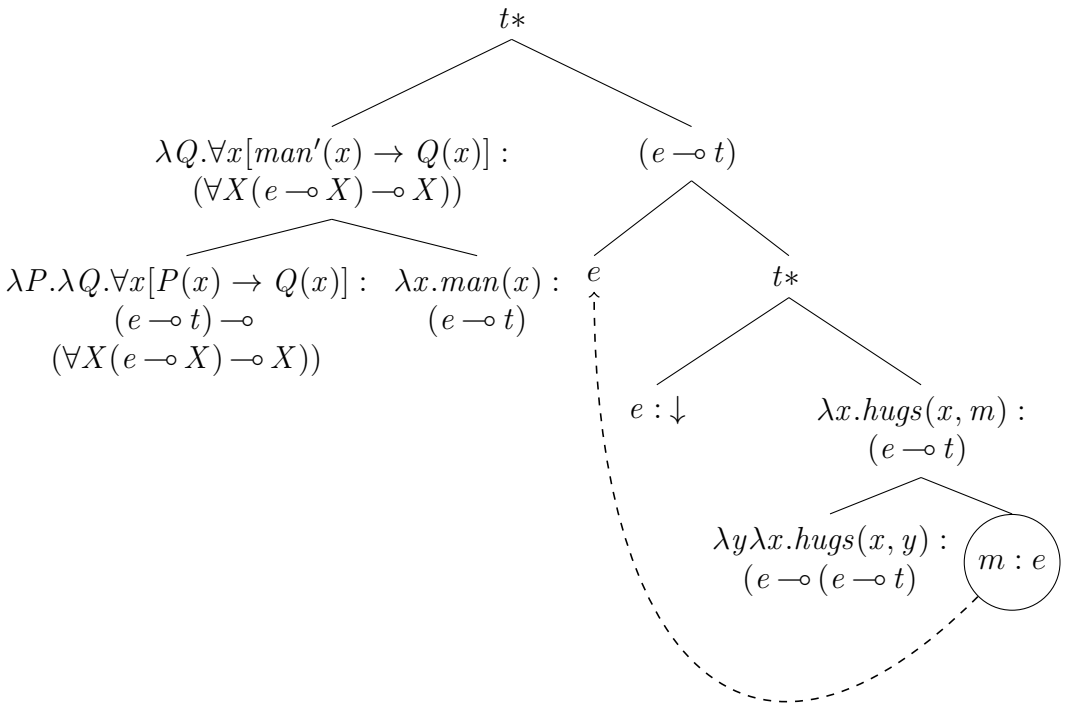
2.4.3 Q scope ambiguities

We now see how our semantics accounts for Qscope ambiguities and for binding. Consider the sentence "Some woman hugged every man". This is ambiguous between the surface ((36a)) and inverse scope ((36b)) readings:

(36) a. $\exists x[woman(x) \wedge \forall y[man(y) \rightarrow hugged(x, y)]]$
b. $\forall y[man(y) \rightarrow \exists x[woman(x) \wedge hugged(x, y)]]$

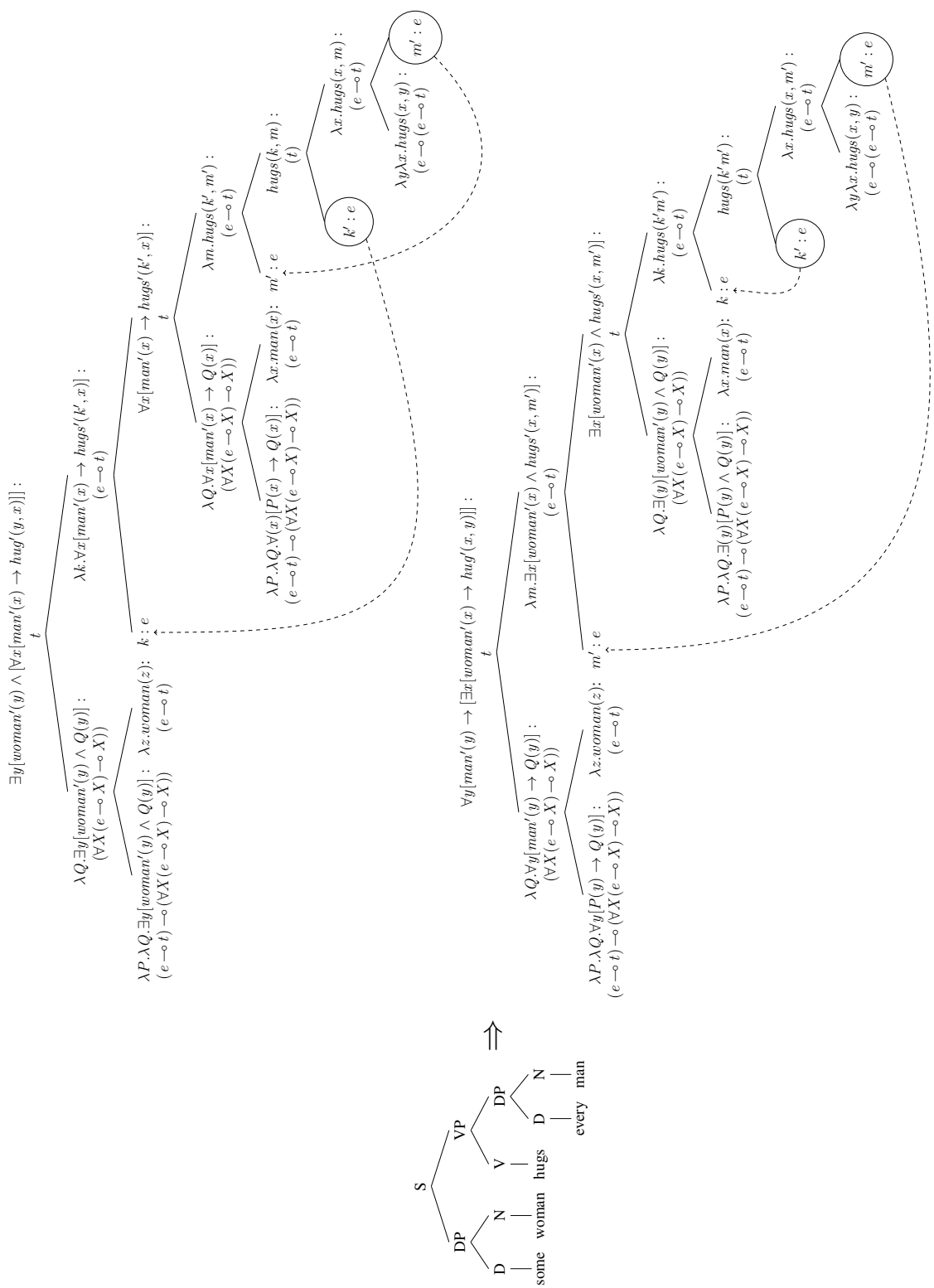
Quantifier scope ambiguities in TAG Glue semantics are generated by quantifiers adjoining to different t nodes in the structure. In the following semantic tree for "hugs every man", you will notice an asterisk on each t node, indicating adjoining sites:

(37)



To derive (36a), a single operation simultaneously adjoins "some woman" to the root node of (37) and substitutes in the hypothesis in the elementary tree set (the range of this operation is the domain of a chain of function which can be composed into a single SNAP function). To derive (36b), a single operation simultaneously adjoins "some woman" to the other t node in (37) and substitutes in the hypothesis in the elementary tree set (the range of this operation is the domain of a chain of function which can be composed into a single SNAP function). This is show on the following page, where squares have been put around each adjunction site:

(38)


S
DP
D
some
N
woman
VP
V
hugs
DP
D
every
N
man
⇑
$\exists y[woman'(y) \wedge [\forall x[man'(x) \rightarrow hug'(y,x)]]] : t$
$\lambda Q.\exists y[woman'(y) \wedge Q(y)] : (\forall X(e \multimap X) \multimap X))$
$\lambda k.\forall x[man'(x) \rightarrow hugs'(k',x)] : (e \multimap t)$
$\lambda P.\lambda Q.\exists y[woman'(y) \wedge Q(y)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda z.woman(z) : (e \multimap t)$
$k : e$
$\forall x[man'(x) \rightarrow hugs'(k',x)] : t$
$\lambda Q.\forall x[man'(x) \rightarrow Q(x)] : (\forall X(e \multimap X) \multimap X))$
$\lambda m.hugs(k',m') : (e \multimap t)$
$\lambda P.\lambda Q.\forall x[P(x) \rightarrow Q(x)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda x.man(x) : (e \multimap t)$
$m' : e$
$hugs(k,m) : (t)$
$k' : e$
$\lambda x.hugs(x,m) : (e \multimap t)$
$\lambda y \lambda x.hugs(x,y) : (e \multimap (e \multimap t))$
$m' : e$
$\forall y[man'(y) \rightarrow [\exists x[woman'(x) \rightarrow hug'(x,y)]]] : t$
$\lambda Q.\forall y[man'(y) \rightarrow Q(y)] : (\forall X(e \multimap X) \multimap X))$
$\lambda m.\exists x[woman'(x) \wedge hugs'(x,m')] : (e \multimap t)$
$\lambda P.\lambda Q.\forall y[P(y) \rightarrow Q(y)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda z.woman(z) : (e \multimap t)$
$m' : e$
$\exists x[woman'(x) \wedge hugs'(x,m')] : t$
$\lambda Q.\exists y[woman'(y) \wedge Q(y)] : (\forall X(e \multimap X) \multimap X))$
$\lambda k.hugs(k',m') : (e \multimap t)$
$\lambda P.\lambda Q.\exists y[P(y) \wedge Q(y)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda x.man(x) : (e \multimap t)$
$k : e$
$hugs(k',m') : (t)$
$k' : e$
$\lambda x.hugs(x,m') : (e \multimap t)$
$\lambda y \lambda x.hugs(x,y) : (e \multimap (e \multimap t))$
$m' : e$

In this way, a single syntactic structure is mapped to two parse trees, which we put in a set. We thus have a simultaneous operation (of SNAP and adjunction) which, applied to the semantic values of the sub-constituents of the syntax tree, produce a set of semantic values for our syntactic structure.

Consider a more complicated example: "Every representative of some company saw a sample". Our informants detect the following 5 readings of this sentence:

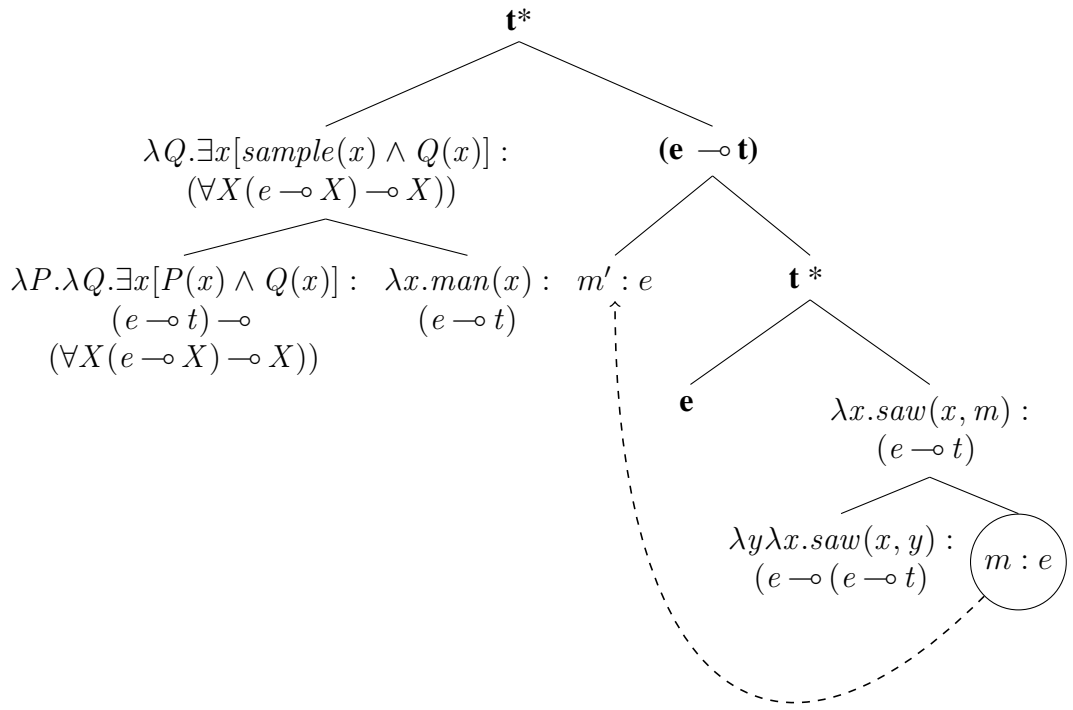
(39) a. $\forall x \exists y[Company(y) \wedge Representative - of(x, y) \rightarrow \exists z[sample(z) \wedge saw(x, z)]$
b. $\exists z[sample(z) \wedge \forall x \exists y[Company(y) \wedge Representative - of(x, y) \rightarrow saw(x, z)]]$
c. $\exists z[company(z) \wedge \forall x \exists y[sample(y) \wedge Representative - of(x, z) \rightarrow saw(x, y)]$
d. $\exists z \exists y[company(z) \wedge sample(y) \wedge \forall x[Representative - of(x, z)) \rightarrow saw(x, y)]$
e. $\exists y \exists z[sample(y) \wedge company(z) \wedge \forall x[Representative - of(x, z)) \rightarrow saw(x, y)]$

The following formula, where an unbound variable y appears in the antecedent, does not correspond to a reading of the sentence and is not derivable in glue (see Asudeh and Crouch (2001b)):

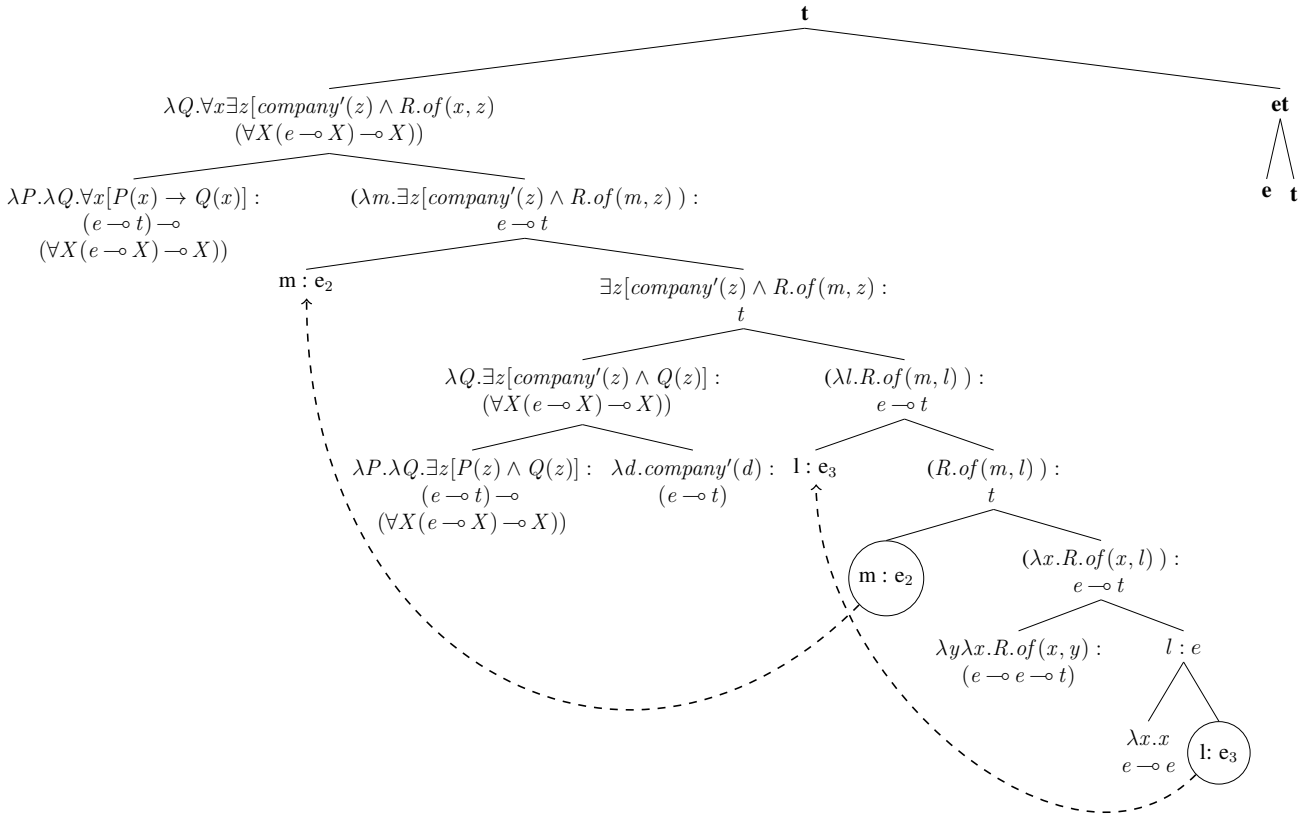
(40) $\forall x[rep - of(x, y) \rightarrow \exists y[company(y) \wedge \exists z sample(z) \wedge see(x, z)]]$

Here are the semantic trees for "saw a sample" and "every representative of a company":

(41)

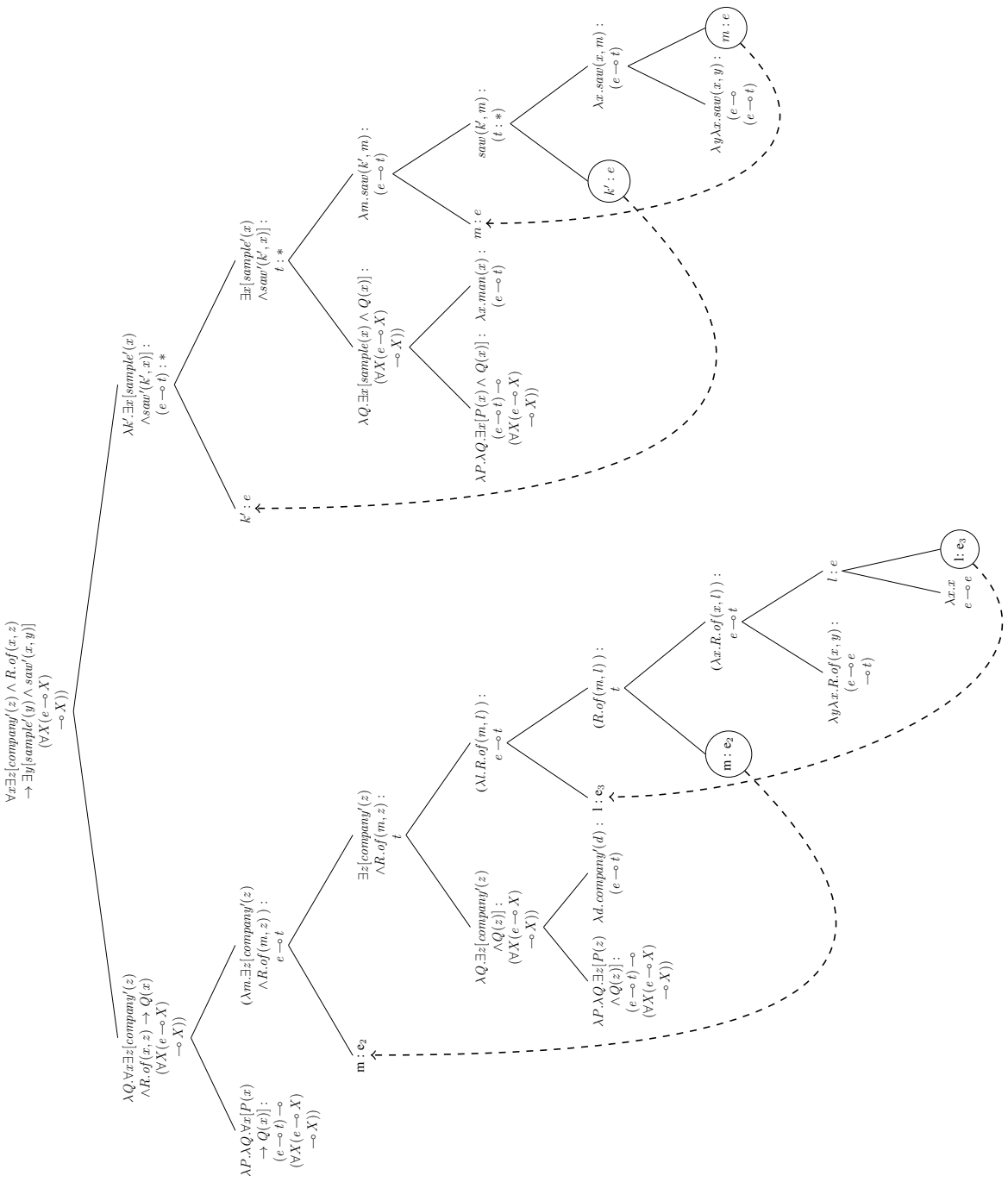

t*
λQ.∃x[sample(x) ∧ Q(x)] :
(∀X(e ⊸ X) ⊸ X))
(e ⊸ t)
λP.λQ.∃x[P(x) ∧ Q(x)] :
(e ⊸ t) ⊸
(∀X(e ⊸ X) ⊸ X))
λx.man(x) :
(e ⊸ t)
m′ : e
t *
e
λx.saw(x, m) :
(e ⊸ t)
λyλx.saw(x, y) :
(e ⊸ (e ⊸ t)
m : e

(42)


t
λQ.∀x∃z[company′(z) ∧ R.of(x, z)
(∀X(e ⊸ X) ⊸ X))
et
e
t
λP.λQ.∀x[P(x) → Q(x)] :
(e ⊸ t) ⊸
(∀X(e ⊸ X) ⊸ X))
(λm.∃z[company′(z) ∧ R.of(m, z)) :
e ⊸ t
m : e2
∃z[company′(z) ∧ R.of(m, z) :
t
λQ.∃z[company′(z) ∧ Q(z)] :
(∀X(e ⊸ X) ⊸ X))
(λl.R.of(m, l)) :
e ⊸ t
λP.λQ.∃z[P(z) ∧ Q(z)] :
(e ⊸ t) ⊸
(∀X(e ⊸ X) ⊸ X))
λd.company′(d) :
(e ⊸ t)
l : e3
(R.of(m, l)) :
t
m : e2
(λx.R.of(x, l)) :
e ⊸ t
λyλx.R.of(x, y) :
(e ⊸ e ⊸ t)
l : e
λx.x
e ⊸ e
l: e3

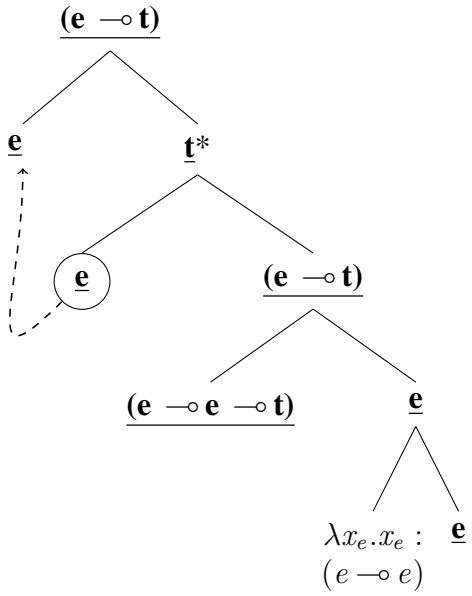
In (41), "a sample" has been adjoined to the root node of the semantic tree for saw and a hypothesis has been substituted in the object position of the semantic tree. We then adjoin (42), the semantic tree for "Every representative of some company", (and substitute in its hypothesis), to derive the full semantic tree:

(43)



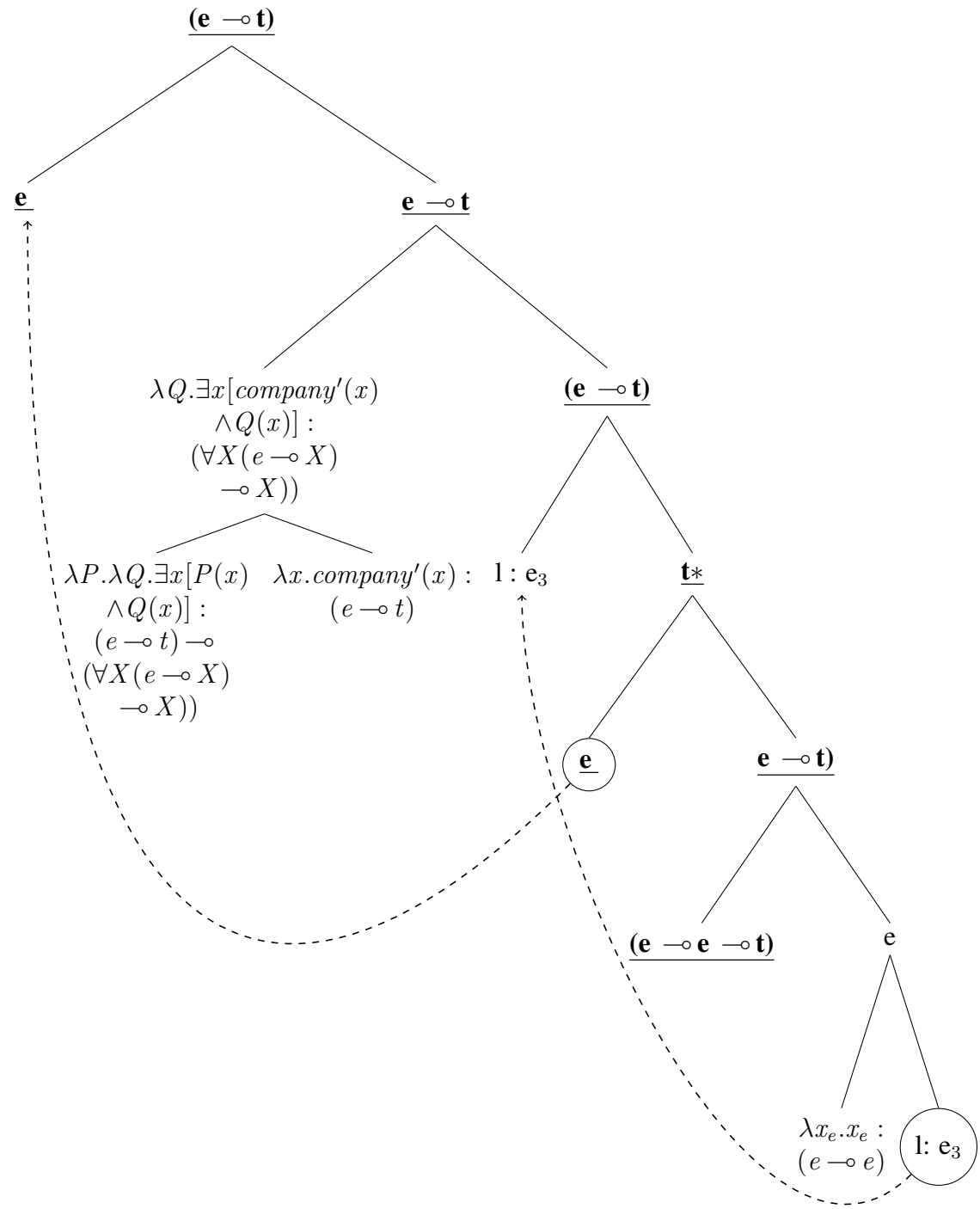
How is *representative of some company* formed? In our system, of has the following semantic tree:

(44)



Of then combines with *some company* by a simultaneous operation of substitution and adjunction to form the following tree:

(45)



We then suppose that relational predicates such as $representative$ have as their denotation elementary tree sets comprising the lexical denotation of $representative$ alongside a hypothesis:

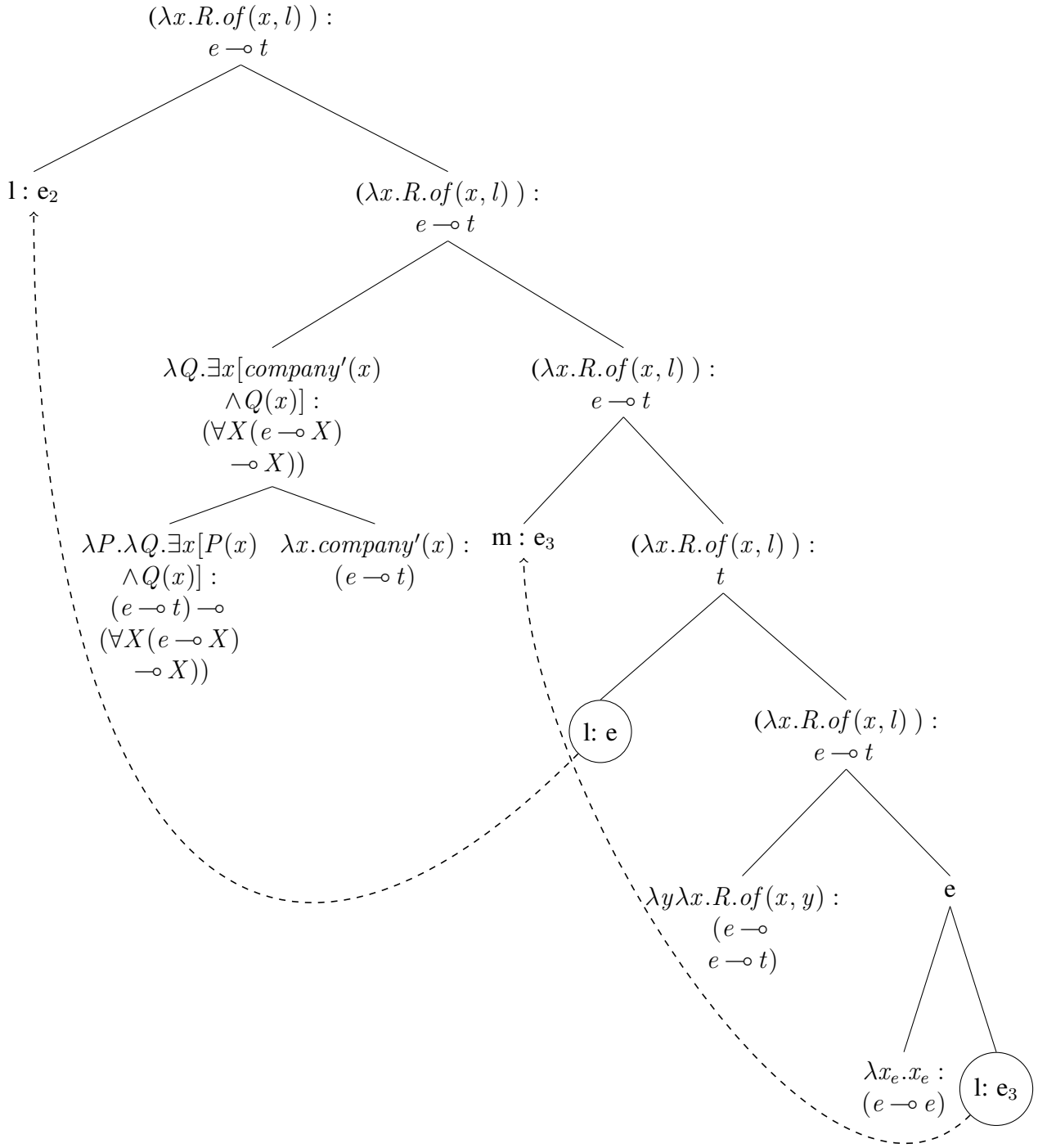
(46)

$$\left\{ \begin{array}{ll} \lambda y.\lambda x.R.of(x,y) & \\ & m : e \end{array} \right\}$$

(2.7)

Both members of this tree set are simultaneously inserted at the **(e $\multimap$ t)** node (in fact, computation of the whole tree could not proceed were they inserted anywhere else):

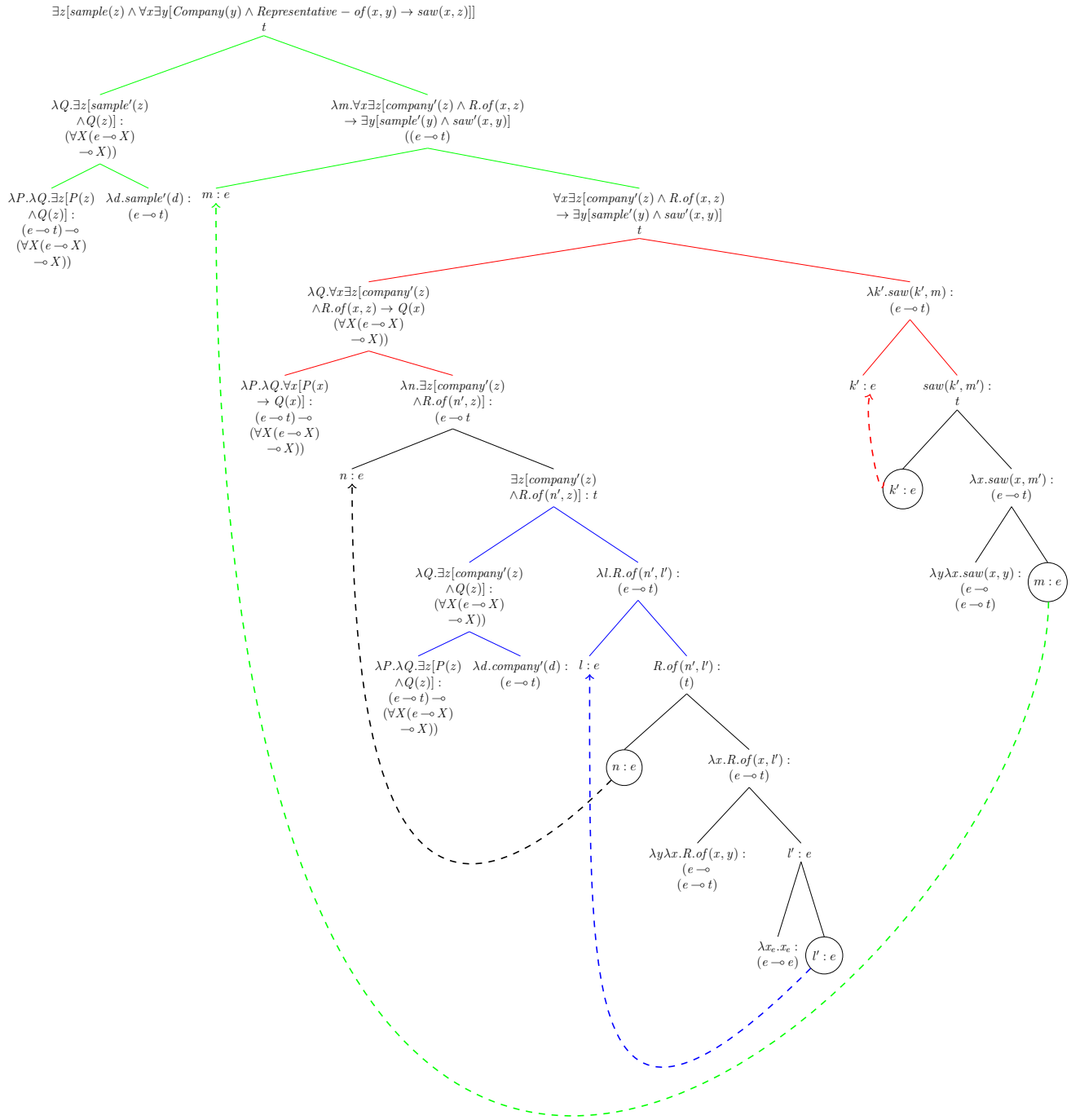
(47)



Every combines with *representative of some company* by simultaneously substituting the root of the latter at $\downarrow$ in the quantifier below and substituting a hypothesis at the remaining variable node in the tree, producing "Every representative of some company" (the tree in (43)).

Reading b. comes about by adjoining "Every representatitive of some company" at the foot node of "a sample" in the semantic tree for "saw some sample":

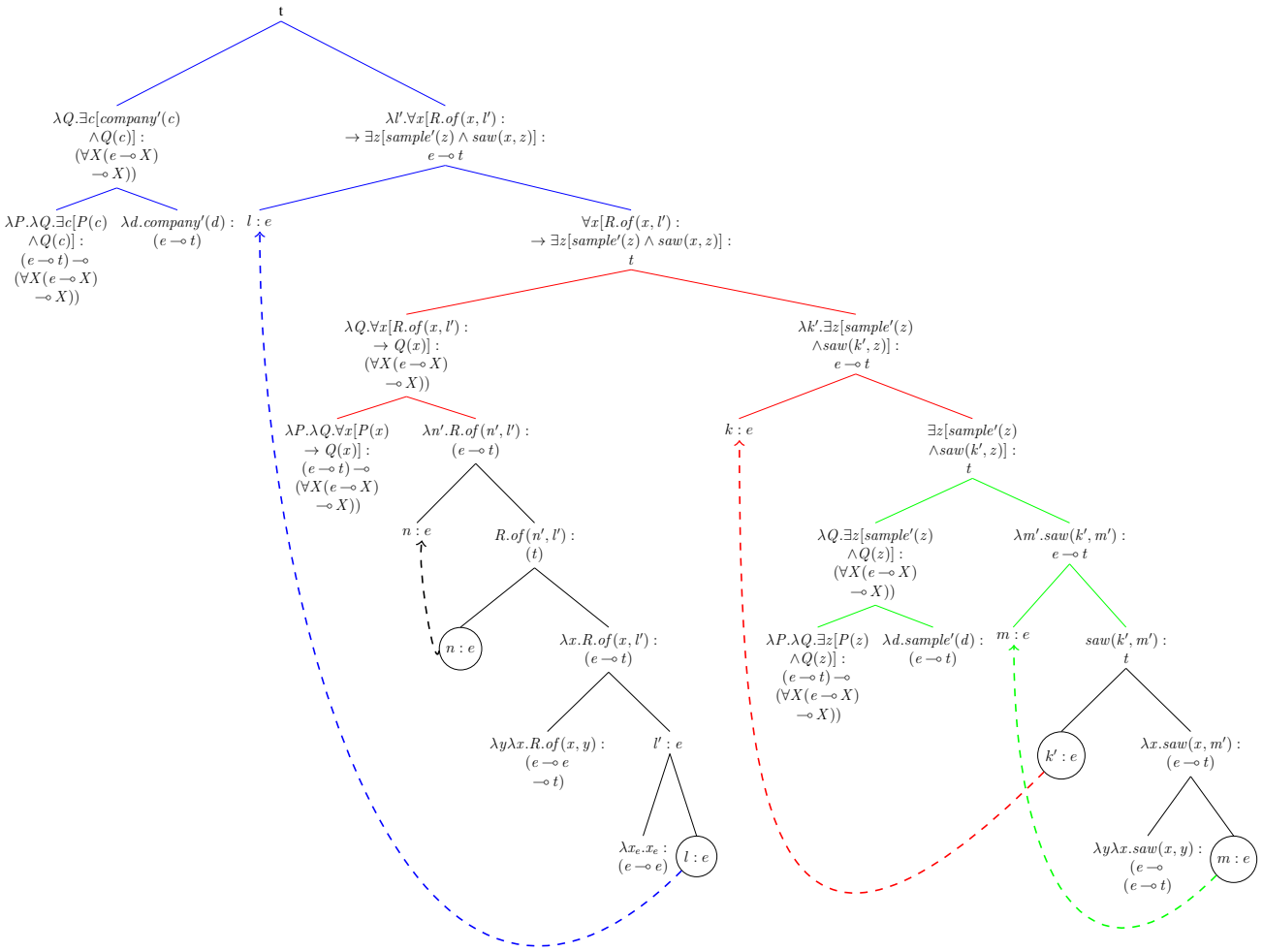
(48)



Reading c. comes about by adjoining "Every representatitive of some company" at the root of the semantic tree for "saw some sample" and then taking "representative of some company" and excising "some company", adjoining it to the root t node of the universal quantifier.

$\exists z[company(z) \wedge \forall x[Representative - of(x,z) \rightarrow \exists ySample(y) \wedge saw(x,y)]]$

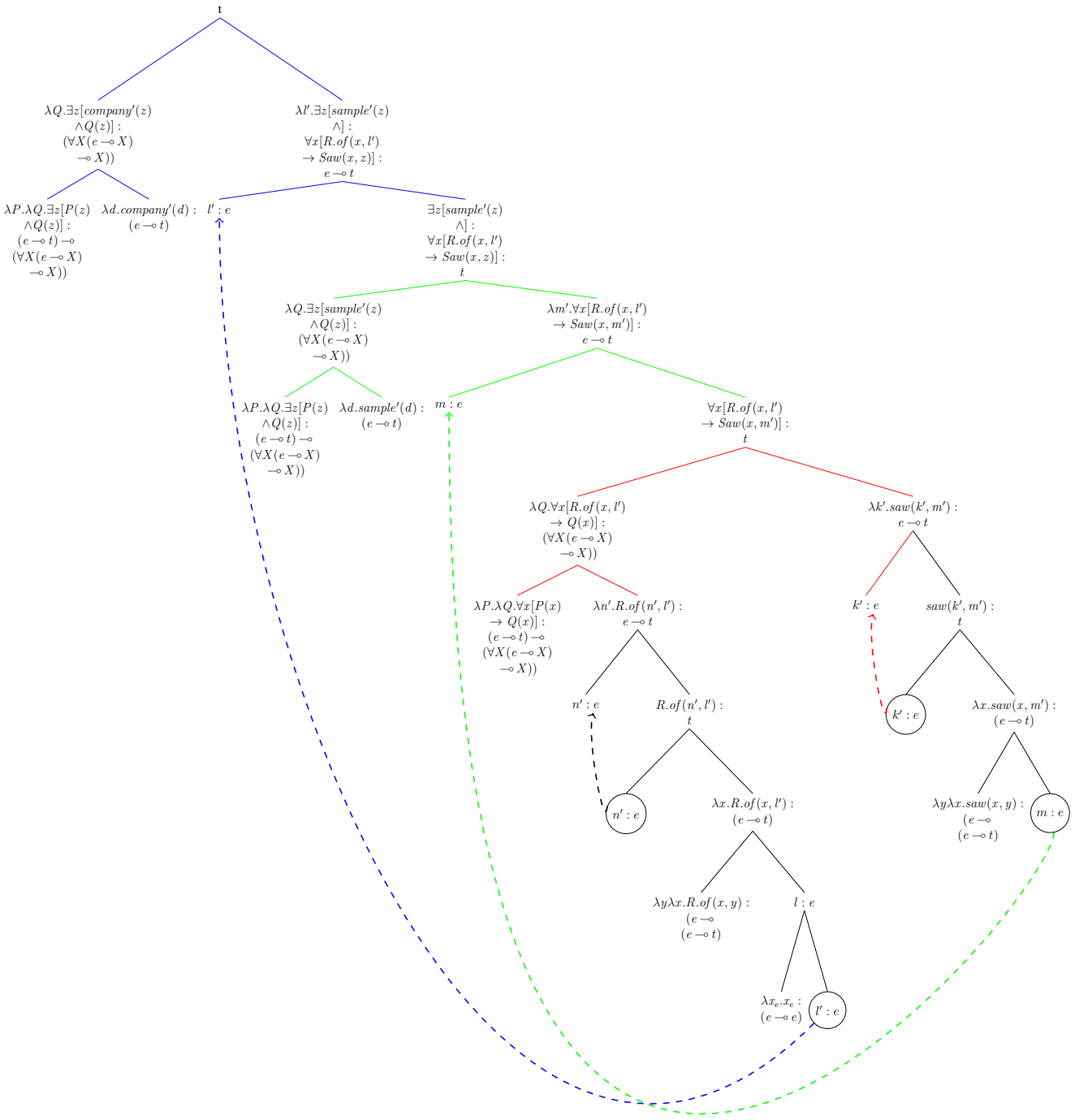
(49)



Reading d. comes about by adjoining "every representative of some company" at the t node in the middle of "saw some company". Then, "some company" is excised from the structure and adjoined to the root t node of "some sample".

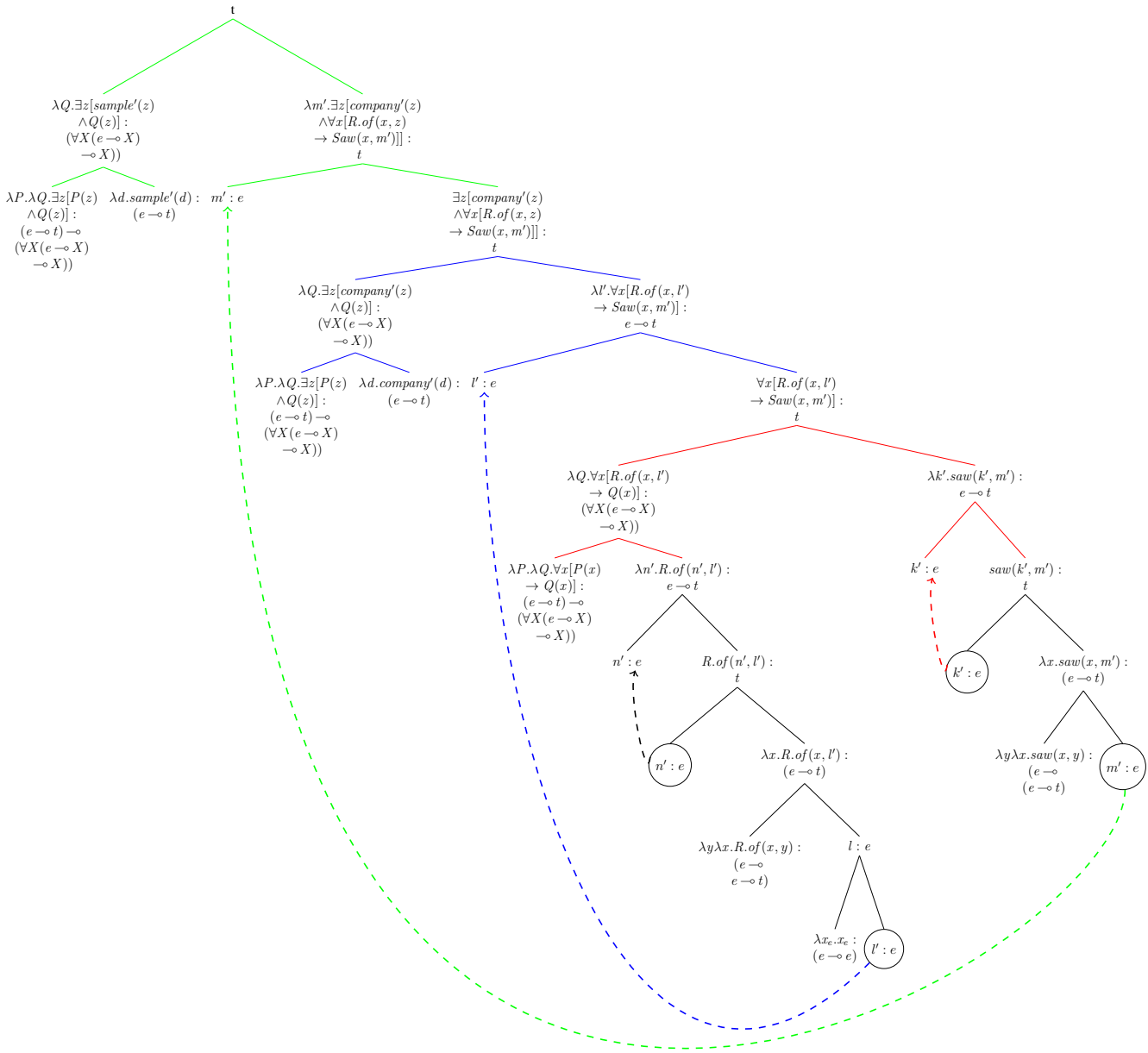
$\exists z \exists y[company(z) \wedge \exists y sample(y) \wedge \forall x[Representative - of(x, z)) \rightarrow saw(x, y)]$

(50)



Finally, reading e. comes about in the same way as e., except by adoining “some company” lower than “some sample”.

(51)



We are now in a position to claim the same advantage for our theory of quantification that Barker claims for his continuation-based approach: that it gives a single unified answer to the question of scope displacement, scope ambiguity and duality of NP meaning identified above. Quantified NPs denote the same kind of object as NPs and thus there is no type clash when quantifiers appear in object position. Scope displacement is achieved merely by the nature of the operation (adjunction) by means of which a verb is combined with a NP. Finally, scope ambiguity is simply a result of the fact that such displacement can happen in different ways. We now turn to the final question in our list: how is variable binding achieved in our system?

2.4.4 Binding

How can the Glue TAG semantics cope with cases in which a quantifer binds a pronoun, such as the following?

(52) Every actor likes himself.

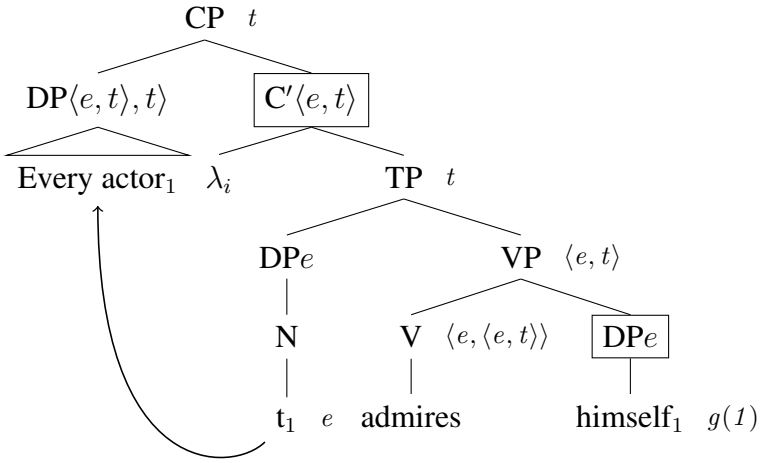
An in depth discussion of binding cannot be undertaken here. But we can underline our general approach. The standard way to deal with such cases in glue is to use a variable free system (see Asudeh). However, the semantic trees produced by such analyses fail to fit transparently onto the type of semantics we have pursued till now, for reasons which we do not have time to discuss.

Our approach to binding is based on Heim and Kratzer (1998), whose proposals can be easily grafted onto our own. In Heim and Kratzer's semantics, quantifiers and the pronouns they bind are co-indexed. Pronouns and traces are interpreted as variables that have a denotation of type e relative to an assignment function, according to the trace and pronouns rule:

If α is a trace or a pronoun, g is a variable assignment, and $i \in dom(g)$, then $[\![\alpha_i]\!]^g = g(i)$.

When a quantifier undergoes QR the rule of predicate abstraction causes the pronoun or any trace with the same index as the quantifier to become bound by the lambda operator created by predicate abstraction, as in the following example:

(53)



Computation then proceeds as follows:

1. $[\![$ VP $]\!] = \lambda y.\lambda x.A(x, y)(g(1))$
2. $\lambda x.A(x, g(1))$ (by $\Rightarrow_\beta$)
3. $[\![$ TP $]\!] = \lambda x.A(x, g(1))\ ([\![t_1]\!]^{g(t_1 \mapsto z)})$
4. $A([\![t_1]\!]^{g(t_1 \mapsto z)}, g(1))$ (by $\Rightarrow_\beta$)

5. $A(z, z)$ (trace and pronoun rule)

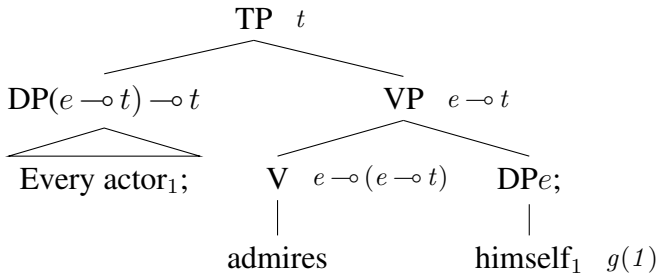
6. $[\![\text{C}']\!] = \lambda z.A(z, z)$

7. $[\![\text{CP}]\!] = \forall x[Actor(x) \rightarrow \lambda z.A(z, z)(x)$

8. $\forall x[Actor(x) \rightarrow A(x, x)$ by $\Rightarrow_\beta$

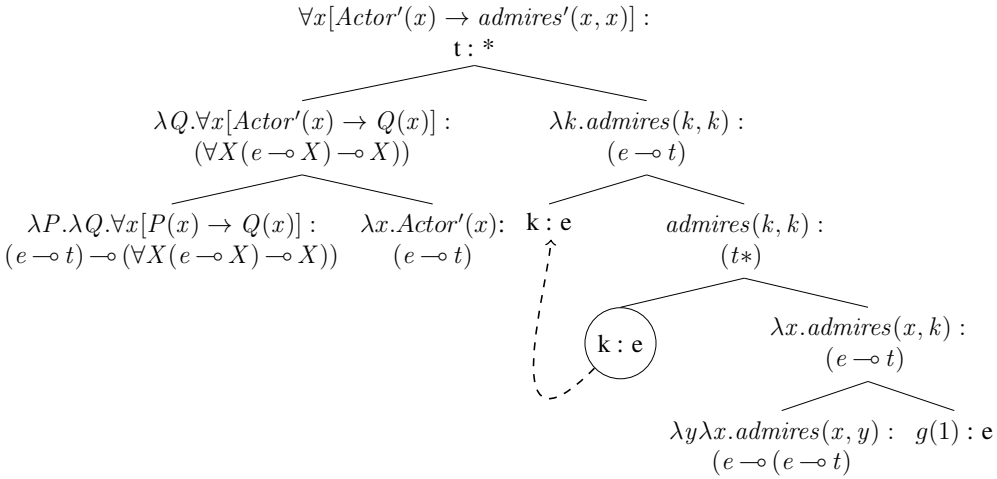
A glue proof semantic tree can be formed which is similar to this LF, except for the fact that no rule governing traces is necessary, since hypotheses in the glue proof play the role of traces in the above LF, and they follow from the underlying linear logic. Suppose we have the following syntax tree, where the reflexive pronoun is coindexed with the quantifier phrase and the core semantic values of each constituent are written:

(54)



We suppose that the hypothesis in the tree set for *every man* has the same denotation as the reflexive pronoun (k), which is interpreted as the result of applying the assignment function to the index associated with it and with the quantifier.

In the glue proof for (54), predicate abstraction abstracts over both the quantifiers hypothesis and the object hypothesis (the assignment function assigns k to the index 1 and the hypothesis in the tree set is k):



This approach may need to be refined in a number of ways. It does however constitute the basis for an account of binding in our framework, and thus answers the question of how variable binding is accounted for in it.

2.4.5 Constraints on adjunction

Scope constraints as restrictions on adjunction and other scope bearing items

Quantifiers do not scope freely. For example, consider (55):

(55) A teacher thinks that every student smokes.

(55) does not have the reading on which for every student x there is some teacher y who thinks x smokes. For this reason we need to propose a mechanism to block the free scoping of quantifiers. A further question is what explains why the quantifiers in a given case are unable to freely scope. We do not answer this explanatory question.

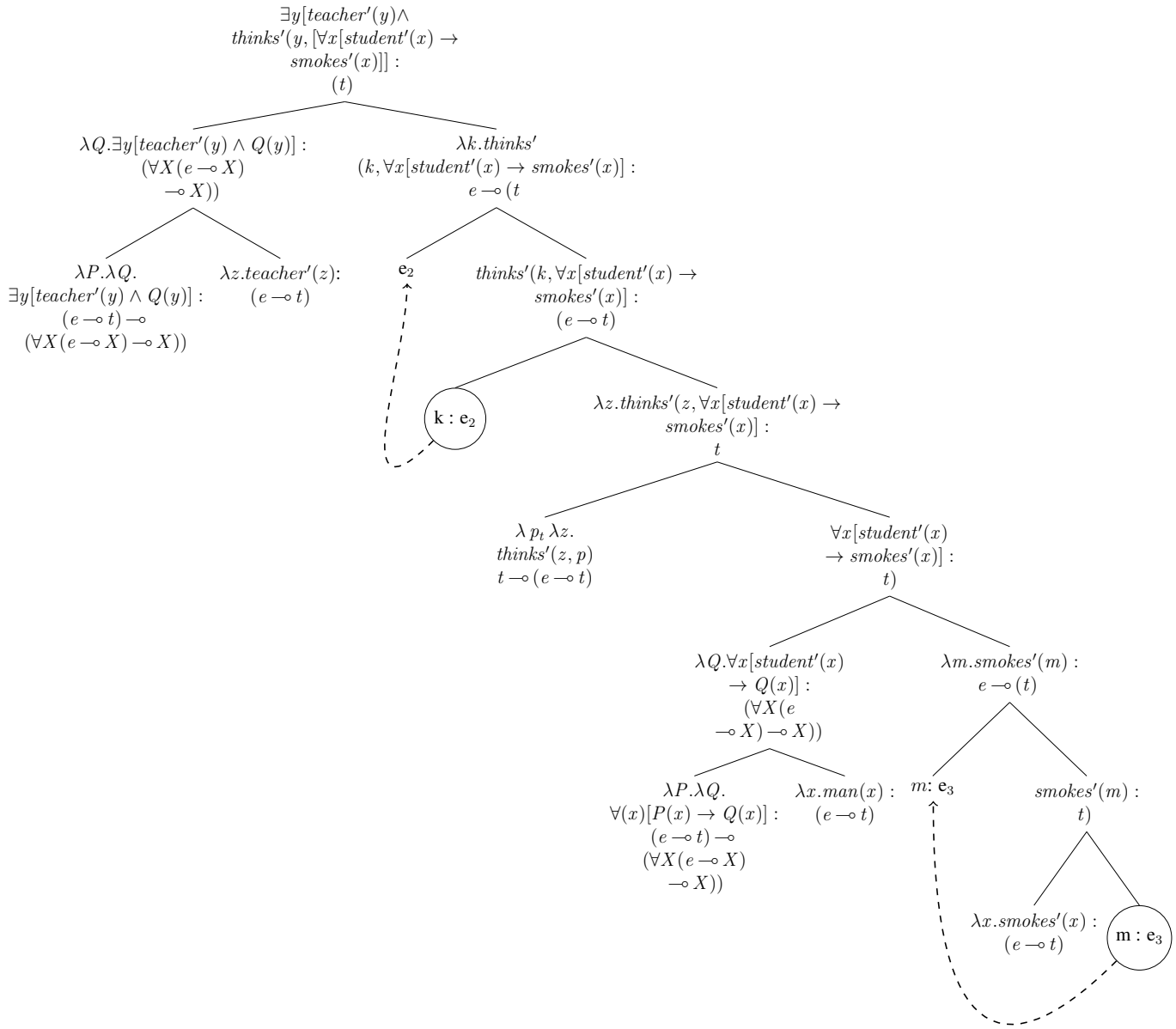
We will consider 3 cases in which quantifier scope is not free. Firstly, quantifiers are not usually able to scope outside of certain so-called syntactic islands. Secondly, there are so-called scope freezing effects, whereby only the surface scope order of a number of quantifiers is available. Thirdly, there are restrictions on whether a quantifier can intervene between the scope of two quantifiers.

Scope islands

Quantifiers do not seem to be able to scope freely outside of tensed clauses, as (55) suggested. Syntacticians have proposed that certain syntactic configurations act as islands, blocking movement of quantifiers outside a given island. QR is thus thought to be restricted by conditions pertaining to the restriction of covert movement. In Glue TAG semantics, unattested quantifier scope relations are regulated similarly, by appropriate adjunction constraints, which prevent quantifiers from adjoining at certain t nodes. This fits in very nicely with the TAG architecture, according to which adjunction constraints may be imposed when necessary (see Joshi and Schabes (1997)). Consider the following semantic tree for (55):[6]

(56)

[6]This has been simplified by assuming an extensional denotation for *think*



In order to prevent the unattested reading we require that the quantifier "Every student" adjoins below "think" in the semantic tree. This prevents the reading where the universal takes wide scope from coming about. Whatever the details of such an account we therefore know that adjoining constraints must be formulated and justified, in order to block such readings. Again, the same proviso applies that we mentioned above: this, and all of our proposals about blocking unattested readings are very tentative and preliminary suggestions. The fact that adjunction constraints in TAG have been used to block unattested readings gives us some measure of hope.

More generally, when considering scope islands, adjoining may be prevented from occurring above a certain node in the semantic tree. The advantage of this account is that it can state restrictions on quantifier scope in terms of restrictions on movement. It can thus potentially emulate certain restrictions on QR that have been proposed by minimalists, the only difference being that the restrictions are stated in the semantics. Thus, at least philosophically speaking, we can agree

with those who pose an intimate link between restrictions on syntactic scope and restrictions on semantic scope restrictions. This is comforting, given that many generalisations about semantic scope have been given syntactic analyses.

Scope freezing effects

So called scope freezing effects arise in a number of situations, amongst which the following ditransitive constructions:

(57) John gave a politician every report $((\exists > \forall), \quad ??(\forall > \exists))$

There is no reading of this sentence where for every report x John gave x to some politician y.

Scope freezing effects in Glue TAG are derived by two principal means. Firstly. we can prevent that the quantifier "every report" adjoins above the t node of "a politician". Secondly, we can postulate that certain groups of quantifiers form a scope block. A scope block is a tree comprising two quantifiers, one adjoined to the other, whose root and foot is a t node. Scope blocks may in certain circumstances be broken up by other scopally relative operators (for example, negation), in which case they are *flexible* scope blocks. However with *inflexible* scope blocks this is not be possible (more on this soon). Sometimes a scope block may be *non-rigid*; that is, the relative order of quantifiers in the block may be changed. However, in other cases a scope block is *rigid*; the relative order of quantifiers in the block is invariant. In order to account for scope freezing effects we therefore propose that "a politician" and "every report" form a rigid scope block. This ensures that the order of the quantifiers in the block cannot be reversed. To be fully convincing our suggestions would need to be complemented by an explanation of why the relevant scope blocks are formed, and why they have the properties that they do. But we do not have the space to go into this.

Larson's generalisation

Larson noted that in sentences with quantifiers embedded in PPs headed by quantifiers, free scoping of the quantifiers is not available. Consider the following sentence:

(58) More than half of the students will investigate [$_{DP}$ at least one dialect [$_{PP}$ of every language]].

(58) does not have an interpretation in which, for each language x, more than half of the students y investigated a dialect of x, but no student investigated the same dialect as any another.

In the Glue TAG semantics, such restrictions are accounted for by treating "at least one dialect" and "every language" as forming an inflexible scope block (an inflexible, non-rigid scope block, to be exact). This prevents a quantifier from being adjoined at the t node inbetween the two quantifiers.

Again, our suggestion needs to be complemented with an explanation of why the relevant scope blocks are formed, and why they have the properties that they do.

Negation

Quantifiers interact scopally with negation, to produce ambiguities. For example, consider the following sentences (in the following examples $a > b$ indicates the reading where a scopes over b and "??" indicates that the given reading is infelicitous):

(59) Every student didn't show up $(\forall > \neg)$ / $(\neg > \forall)$

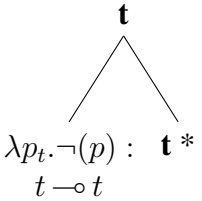
(60) More than three students didn't show up. $(\exists > \neg)$ / $??(\neg > \exists)$

(61) John didn't meet every student $(\neg > \forall)$ / $??(\forall > \neg)$

(62) John didn't meet more than three students $(Q_{>3} > \neg)$ / $(\neg > Q_{>3})$

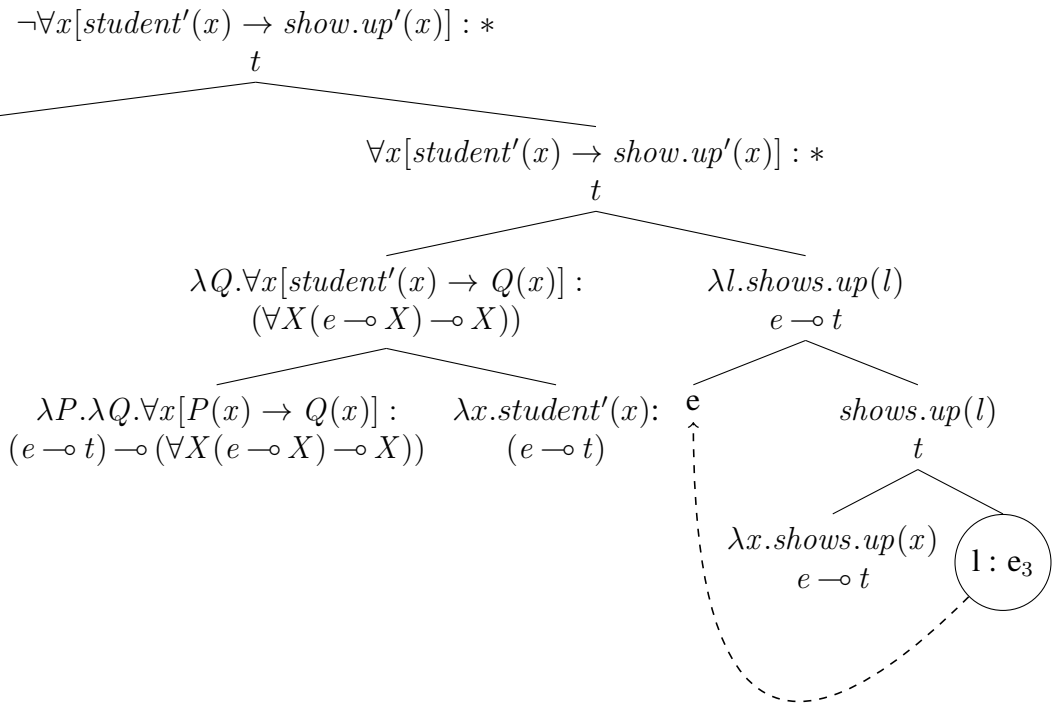
An adequate semantics must have the means to restrict the scope negation in certain cases. In Gl-TAG we treat the semantic value of negation as an auxiliary tree, of the following form:

(63)

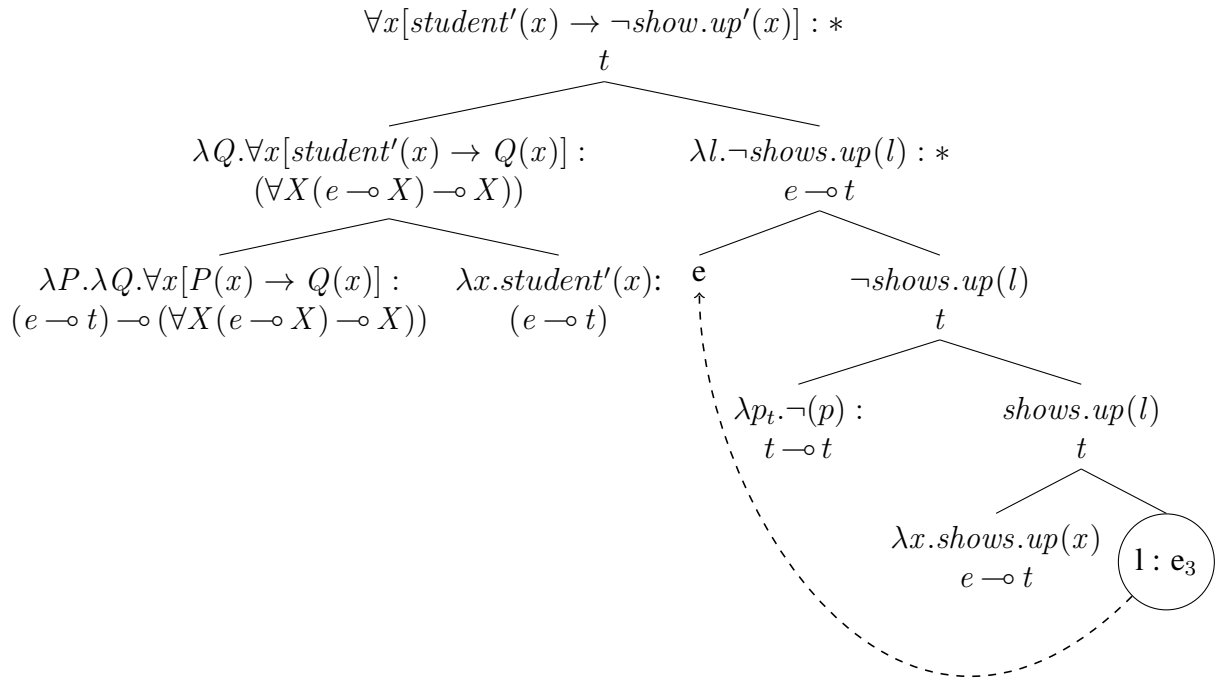


There are three nice feature of this analysis. Firstly, since quantifiers and negation are scope bearing items they bear certain semantic features in common. This is brought out in our analysis by the fact that both are auxiliary tree. Secondly, our analysis allows a uniform treatment of negation, at least in cases such as (59), whose two readings are represented by the following semantic trees:

(64)


$\neg\forall x[student'(x) \rightarrow show.up'(x)] : *$
t
$\lambda p_t.\neg(p) :$
$t \multimap t$
$\forall x[student'(x) \rightarrow show.up'(x)] : *$
t
$\lambda Q.\forall x[student'(x) \rightarrow Q(x)] :$
$(\forall X(e \multimap X) \multimap X))$
$\lambda l.shows.up(l)$
$e \multimap t$
$\lambda P.\lambda Q.\forall x[P(x) \rightarrow Q(x)] :$
$(e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda x.student'(x)$:
$(e \multimap t)$
e
$shows.up(l)$
t
$\lambda x.shows.up(x)$
$e \multimap t$
1 : e$_3$

(65)


$\forall x[student'(x) \rightarrow \neg show.up'(x)] : *$
t
$\lambda Q.\forall x[student'(x) \rightarrow Q(x)] :$
$(\forall X(e \multimap X) \multimap X))$
$\lambda l.\neg shows.up(l) : *$
$e \multimap t$
$\lambda P.\lambda Q.\forall x[P(x) \rightarrow Q(x)] :$
$(e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda x.student'(x)$:
$(e \multimap t)$
e
$\neg shows.up(l)$
t
$\lambda p_t.\neg(p) :$
$t \multimap t$
$shows.up(l)$
t
$\lambda x.shows.up(x)$
$e \multimap t$
1 : e$_3$

As can be seen, predicate negation and sentential negation both receive a uniform analysis. This allows for the rules for negation to be somewhat more simple than they would otherwise be.

Thirdly, and finally, since negation is represented by an auxiliary tree recursive on t, we can restrict the scope of negation by restricting the nodes to which it can adjoin. This allows us to invent certain adjoining constraints that prevent negation from scoping high. Thus, our account allows, in principle at least, for restrictions on the scope of negation, although, we repeat, our suggestion needs to be complemented with an explanation of why the relevant scope restrictions arise, and why they have the properties that they do.

2.4.6 Conclusion

We have shown that TAG Glue semantics offers a novel theory of quantification and can deal with a fragment involving simple transitive and intransitive sentences. Our approach involves making a distinction between the core and peripheral semantic value of an expression, which we argued was motivated and explanatory. In the next chapter we will explore some of the interesting properties of our semantics in comparison to the approaches outlined in the first chapter.

Chapter 3

Gl-TAG and compositi

3.1 Some properties of my semantics

3.2 Compositionality

3.2.1 Intro

In this chapter we show that our TAG Glue semantics is compositional. We first provide an overview of technical work on compositionality, distinguishing versions of the principle and strengthening them. We then discuss two formulations of compositionality, Jonnsen compositionality and Westerståhl compositionality, which permit an analysis of Qscope ambiguous sentences as having a single syntactic structure that is paired with more than one semantic interpretation. We then show that TAG Glue semantics is compositional in both formulations. We finish the chapter with some brief reflections on the similarities between our semantics for Qscope ambiguities and those discussed in chapter 1.

3.2.2 Montague-Hendriks-Jannsen

The principle of compostiionality (POC) is a semantic principle widely assumed to be true and to play a number of explanatory roles (see Jönsson (2008), Pelletier (1999) for critical discussion of these issues). The principle can be formulated as follows:

(POC) The semantic value of a complex expression $\alpha(e_1, ..., e_n)$ is a function only of the semantic values of its immediate constituents $(e_1, ..., e_n)$ and the syntactic mode in which they are combined.

On the rule-to-rule hypothesis Bach (1976), the principle states that for every syntactic rule combining some immediate constituents there is a semantic rule, combining their semantic values. The MHJ (Montague-Hendriks-Jannsen) approach[1] assigns the hypothesis a precise interpretation: the hypothesis entails the syntax and semantics, when represented respectively as many-sorted

[1]There are however a number of differences between the approaches of Jannsen, Hendriks and Montague

algebras S_1 and S_2, are related in the following way; there is a function which associates the operators of the two algebras and there is a homomorphism from S_1 to S_2.

Let the syntax be a many sorted π-algebra $A = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ (The carrier set A is the set of surface expressions) with generating family $H = (H_s)_{s\in S}$ and the semantics be a many sorted ω-algebra $B = \langle (B_t)_{t\in T}, (G_\delta)_{\delta\in\Delta}\rangle$ (see the technical preliminary for relevant definitions),[2] [3] and let $\sigma : S \mapsto T$ and $\rho : \Gamma \mapsto \Delta$ be functions. Then if B is compositional with respect to A the following two conditions hold:

(1) A is (σ, ρ)-*interpretable in B* iff for all $\gamma \in \Gamma$: if $\pi(\gamma) = \langle\langle s_1, ..., s_n\rangle, s_{n+1}\rangle$, then $\omega(\rho(\gamma)) = \langle\langle\sigma(s_1), ..., \sigma(s_n)\rangle, \sigma(s_{n+1})\rangle$.

Suppose A is (σ, ρ)-*interpretable in B* and let $h[A_s]$, for some homomorphism h, denote $\{h(a) \mid a \in A_{sn}\}$. Then:

(2) $h : \bigcup_{s\in S} A_s \mapsto \bigcup_{t\in T} B_t$ is a (σ, ρ)-*homomorphism from A to B* iff

- for all $s \in S : h[A_s] \subseteq B_{\sigma(s)}$
- if $\pi(\gamma) = \langle\langle S_1, ..., s_n\rangle, s_{n+1}\rangle$ and $a_1 \in A_{s\,1}, ..., a_n \in A_{s\,n}$, then $h(F_\gamma(a_1, ..., a_n)) = G_{\rho(\gamma)}(h(a_1), ..., h(a_n)$

However, thus far, since A is the set of surface expressions, structurally distinct expressions receive the same interpretation. Therefore a (σ, ρ)-homomorphism must also hold between an fully disambiguated algebra and the semantic algebra, where the fully disambiguated algebra represents the immediate constituents of a given expression and their structural features. One such algebra is the term algebra of A with respect to generating set H, $T_{A,H} = \langle\ (T_{A,H,s})_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\ \rangle$ (see the technical preliminary). In the term algebra, a constituent is simply a term and an immediate constituent t' of a term t is simply a proper constituent of t such that there is no proper constituent t'' of t such that t' is a proper constituent of t''. The grammatical structure of a term t is simply the operation γ in the term algebra which, applied to $t's$ immediate constituents, produces t (γ is nullary for atomic terms).

(Hendriks (2001) and Janssen (1983) point out that a term τ is usually associated with a translation $tr(\tau)$ into some logic which in turn receives an interpretation with respect to the semantic algebra $I(tr(\tau))$. For this reason, the composition $tr \circ I$ of the translation and the interpretation function must in addition constitute a homomorphism. Furthermore, meaning postulates are often used to ensure certain terms are always assigned a specific interpretation, and these reduce the space of models available for the logical language, making the definition of compositionality more complicated. Hendriks and Jannsen end up imposing quite specific requirements on a semantics in order that it be compositional with respect to a given syntax. Such complexity give rise to a potential problem of applicability, as Partee remarks:

[2]The algebras A and B contain families of sorts, $(A_s)_s \in S$, $(B_t)_t \in T$ respectively in order to represent the fact that expressions may belong to different syntactic categories which are paired with different semantic categories in the semantic algebra. The operators of the two algebras are also a family of sorts, $(F_\gamma)_\gamma \in \Gamma$, $(G_\delta)_\delta \in \Delta\rangle$ respectively and the operations of $(F_\gamma)_\gamma \in \Gamma$ apply to expressions of a given syntactic category and may concatenate, insert, permute or delete material.

[3]Montague instead worked with a more complicated one-sorted algebra

"The relation between a linguist's syntactic component and syntax as an algebra is not always easy to see, and it can be non-trivial to determine whether and how a given syntax can be presented as an algebra, and more particularly,... as a homomorphism to a corresponding semantic algebra."

A simpler, more general formulation of compositionality (Hodges (2001)) has been formulated in recent years. This formulation if the operations of the syntax are restricted in certain ways (Pagin and Westerståhl (2010)), defines a directly compositional semantics in the sense of (Barker and Jacobson (2007)), where a directly compositional semantics requires that for every syntactic rule there is a semantic rule and thus that every constituent is assigned an interpretation. The formulation also makes minimal assumptions about what meanings are, assuming only that meanings form a set; likewise, minimal assumptions are made also about the operations in the syntax and the strings they derive (unless a directly compositional semantics is desired).

Hodges' framework can help determine whether a given semantics is compositional or not more easily than MHJ's framework; it can also be applied to many different syntactic and semantic frameworks without requiring some of the encumbering assumptions of the MHJ framework. To give one example, MHJ require that the semantics be an algebra structured similarly to the syntax, whereas (Hodges (2001)) does not. Whether or not it is desirable to define an algebra in this way, (POC) itself is neutral about whether meanings are so structured: it says only that there is a meaning assigning function for any syntactic operator. Since Hodge's framework, to which we now turn, avoids imposing too many restrictions on semantic and syntactic frameworks in order that they be compositional, it is in a better position to capture the essential features of compositionality more elegantly than the baroque technical formalities of MHJ.

3.2.3 Hodges (2001)

For Hodges (2001) a grammar is a partial algebra, $\mathbf{E} = (E, A, \Sigma)$, where Σ is a set of partial functions over E, a set of (possibly ambiguous) expressions of finite arity which are generated from a set of basic expressions, $A \in E$.[4]

The grammatical term algebra $GT(E, A, \Sigma)$ of the grammar (E, A, Σ) assigns possibly different structures to the same surface expressions, and is defined as the smallest set such that:

1. Every atomic term t is in GT and $val(t)$ is t.

2. For any $\alpha_n \in \Sigma$ the terms $t_0, ..., t_{n-1} \in GT$ and if $val(t_i) = e_i$ for all $t_0, ..., t_{n-1}$ and $\alpha(e_0, ..., e_{n-1})$ is defined, then "$\alpha(t_0, ..., t_{n-1}$" $\in GT$ and $val($"$\alpha(t_0, ..., t_{n-1}$" $= \alpha(e_0, ..., e_{n-1}$.

By 2, val is a surjective homomorphic map from the grammatical terms to expressions in E, which implies that, for a complex term $V(\overline{\alpha}(t_1, ..., t_n))$ we have (where $\overline{\alpha}$ names the relevant syntactic constructor, α in the grammatical term algebra)

$$V(\overline{\alpha}(t_1, ..., t_n)) = \alpha(V(t_1), ..., V(t_n))$$

The interpretation function for a grammar E is a (partial) map, μ, from terms to their semantic values, whose domain is a subset of the GT of E.

[4] Alternatively, by standard algebraic convention we define a grammar E as a partial algebra $\mathbf{E} = (E, \alpha^E)_{\alpha \in \Sigma}$ where E is the set of expressions and the $(\alpha_i) \in \Sigma$ are n-ary partial functions on E (for some $n \geq 0$), Atomic expressions being zero-place functions.

If $\mu(\phi)$ is defined ϕ is $\mu - meaningful$ [5]

Hodge, defining *immediate constituent* and *mode of combination* as in the MHJ approach above,[6] then defines compositionality in the following two ways, which Hodge proves are equivalent if every constituent of an expression is μ-meaningful:

Hodgean compositionality 1
$Funct(\mu)$:

For every syntactic rule $\alpha \in \Sigma$ there is a meaning operation r_α such that if $\alpha(u_1, ..., u_n)$ is meaningful and $(u_1, ..., u_n)$ are immediate constituents of $\alpha \in \Sigma$, then $\mu(\alpha(u_1, ..., u_n)) = r_\alpha(\mu(u_1), ..., \mu(u_n))$.

The right hand side of the equation in Funct presupposes that every sub-constituent of $\alpha(u_1, ..., u_n)$ is μ-meaningful. Hodge's second formulation of compositionality does not assume this is the case, and is designed to render precise the idea that a complex expression's meaning is invariant under certain substitutions of its constituents.

Hodgean compositionality 2
(Subst)

If $\phi[u_1, ..., u_n]$ and $\phi[t_1, ..., t_n]$ are both meaningful expressions (where the bracketing $[u_1, ..., u_n]$ indicates possibly disjoint occurrences of the (possibly not immediate) subterms $[u_1, ..., u_n]$ and if $u_i \equiv_\mu t_i$ for $1 \leq i \leq n$, then $\phi[u_1, ..., u_n] \equiv_\mu \phi[t_1, ..., t_n]$.

Let us now consider whether (Funct) requires strengthening.

3.2.4 Strengthening compositionality

These formal definitions of compositionality tell us that the meaning of a complex expression e is a function of the meaning of its immediate constituents but they do not tell us what kind of function.

In particular, they do not ensure that an average human can compute the function from the semantic values of $e's$ parts, which is desirable if the semantics is to underpin a theory of our language use:

[5] In order to give a relatively theory neutral characterisation of compositionality, Hodge makes no particular assumption about meanings, other than that a relation of synonymy can be defined between them, as follows:
(Where $\equiv$ is a partial equivalence relation between terms), $u \equiv_\mu t$ *iff* $\mu(u), \mu(t)$ are both defined and $\mu(u) = \mu(t)$.
We can then say that two semantics, μ, υ for a grammar are equivalent if their synonymy relations are identical: if $\equiv_\mu = \equiv_\upsilon$.

[6] We can also define syntactic categories from this notion of constituency, by taking a syntactic category X to be an equivalence class ($\sim_X$) of terms with respect to the relation of substitutability:

For $X \subseteq GT$, $t \sim_X u$ iff all terms $s[t]$, $s[t] \in X \leftrightarrow s[u] \in X$ (where $s[t]$ indicates a subterm occurrence of t in s, and su results from replacing it by u.
Semantic categories can also be defined as equivalence classes with respect to preservation of meaningfulness under substitutability, by restricting σ_X to $dom(\mu)$.

> "Strictly speaking, that compositionality holds is not enough for making sure that an interpreter can figure out the meaning of a complex expression. It is not enough that the meaning of a complex expression is a function of the meaning of the parts and the mode of composition, for the function need not be computable. If it is not, the interpreter cannot work out what the value of the function is for new arguments"

A possible solution to this problem is to claim that speakers derive the meaning of an expression by a recursive clause which specifies the value of the function with respect to both atomic elements of the semantics and complex semantic values. A structured semantics of this kind is an algebra, $M = (M, B, \omega)$, comprising of a set of partial $C*$ functions (ω) whose domain is a finite set of basic meanings (B) and whose range is a derived set of meanings, M. Suppose we have a precise idea of what a suitable computable compositional function is (for suggestions see Pagin (2012)). Call such a function a $C* - function/operation$. By recursion over M we have:

Rec(μ^1) There is a $C*$ function $b \in \omega$ and for every $\alpha \in \Sigma$ a $C*$ operation $r_\alpha \in \omega$ such that for every meaningful expression s,

$$\mu(s) = \begin{cases} b(s) & if\ s\ is\ atomic \\ r_\alpha(\mu(u_1), ..., \mu(u_n), u_1, ..., u_n) & if\ s\ = \alpha(u_1, ..., u_n) \end{cases}$$

The modesty of Hodgean compositionality is also evident from the fact that it does not demand that the semantic value of one daughter of every node is a function and the semantic value of its sister an argument; such a requirement enables us to reduce redundancy in the semantics since the denotation of each node in the tree is predictable from the type of its daughters (the requirement that the semantics be *type driven*). However, hodgean compositionality is compatible with there being construction specific compositional rules.

3.3 Compositionality and Qscope ambiguities

Funct(μ) presupposes two claims Jönsson (2008). According to the *determination claim*, the semantic value of a complex expression is determined by the semantic values of its immediate constituents and their syntactic combination. According to the *quantified claim* a complex expression has a unique value relative to a structural analysis. But (Subst), the more general version of compositionality, does not seem to entail the quantified claim.[7]. Jonnsen argues that the quantified claim is not necessary in order for POC to discharge its explanatory duty (Jönsson 2008:30). Since he seeks the minimal formulation of POC which serves this purpose, he rejects the quantified claim, and points out two further difficulties with it.

Firstly, the quantified claim forecloses an open empirical matter. It is surely better to leave it open in principle that a part of syntactic structure may receive no semantic interpretation. Although it may be a working assumption to assign meaning to every part of constituent structure, this does not imply that a semantic theory which does not assume this is mistaken from the outset, unless

[7]Westerståhl points out that, relative to certain definitions of synonymity, (Subst) also requires each term have a unique meaning: "(Subst) does [suppose expressions have a unique semantic value] too, in a less perspicuous way, in that it uses a straightforward notion of synonymy: two expressions are synonymous if (both are meaningful and) they have *the same meaning*."

there is evidence that the theory assigns no meaning to parts of syntactic structure which intuitively have meaning.

Secondly, the quantified claim requires that a syntactically unambiguous sentence can only have two interpretations if we can form a set of its interpretations (a set-valued function from the syntax to the semantics). We analysed Qscope ambiguities as a function from a single syntactic structure to a set of parse trees of glue proofs (Cooper storage and the flexible types approach to Q scope ambiguities also adopt this option). But we may wish to see whether a version of compositionality can be adopted which does not depend on there being such a set-valued function between the syntax and the semantics; and, in any case, we wish to see whether our semantics allows such a set-valued function. In the following two subsections, we explore two revisions of Hodges' version of compositionality: the first pairs a syntactic structure with a set of interpretations; the second formulates a relational version of compositionality. We then see if our semantics comes out compositional on these versions.

3.3.1 Approach 1: Jonnsen compositionality

Jonnsen's revision of Hodge's version of compositionality introduces the notion of a *semantic range*, "the possibly empty set of meanings of a grammatical term" (Jönsson 2008:31). The interpretation function μ is a total function, assigning semantic ranges to grammatical terms, drawn from the power set of meanings, such that for grammatical term p, we have:

Definition 21 $\mu(p) \subseteq \mathcal{P}\{\mu(x) : x \textit{ is a GT in the domain of } \mu\}$

Derivately, the semantic value of an expression e is the (possibly infinite) union of the semantic ranges of its structural analyses (for i-many n-ary (n $\geq$ 0) syntactic rules σ_i^n):

Definition 22 $\mu(e) = \bigcup_{i=1}^{\infty} \mu(\sigma_i(e)))$

On this picture, compositionality is defined as follows:

Definition 23 A semantic theory is Jonssen compositional iff

For every complex grammatical term $s = \sigma(e_0, ..., e_{n-1})$, there is a function r such that $\mu(s) = r(\sigma, \mu(e_0), ..., \mu(e_{n-1}))$.

Call a semantic theory *thinly compositional* if it is Jonssen compositional and does not require that a grammatical term have a non-empty, unique semantic range. Call a semantic theory *thickly compositional* which is Jonssen compositional and requires that a grammatical term have a unique semantic range.

A thinly compositional semantic theory allows that syntactic rules may be associated with more than one semantic rule of combination. A single syntactic structure can then be associated with a semantic range ≥ 1. A thickly compositional theory cannot allow this, since a grammatical term must have a unique semantic range. Is the semantics we have proposed thinly compositional? Let us go through the types of sentence found in our fragment.

First off we have intransitive sentences with proper names as subjects. Let σ be the syntactic operation that merges a DP subject with an intransitive VP. So we have that

(3) $\sigma(e_{DP}, e_{VP})$

By Jonnsen compositionality, there must be some r such that

(4) $\mu(\sigma(e_{DP}, e_{VP}) = r(\sigma, (\mu e_{DP}, \mu e_{VP})))$

In our system (in which proper names are quantifiers) $\mu(e_{DP})$ is an auxiliary tree corresponding to a quantifier which combines with a tree corresponding to the VP by a simultaneous operation on the tree set for the quantifier that adjoins one of its members (the auxiliary tree) to the VP and substitutes the other member in at the variable node in the VP semantic tree. Thus there is a function, r, of simultaneous adjunction and substitution, that satisfies the requirements of compositionality.

For a simple transitive sentence Let σ_0 be the syntactic operation that combines the verb with its direct object and σ_1 be the syntactic operation that combines the subject with the VP. Then we have

(5) $\sigma(e_V, e_{DP})$

(6) $\sigma(e_{DP}, e_{VP})$

Regarding (5), by Jonnsen compositionality, there must be some r such that

(7) $\mu(\sigma(e_V, e_{DP})) = r(\sigma, (\mu e_V, \mu e_{DP}))$

Again, there is such an r, namely, simultaneous adjunction and substitution. We adjoin the semantic tree for quantifier DP to the semantic tree for the verb and substitute in its hypothesis. As regards (6) the relevant function r is (again) the operation of simultaneous adjunction and substitution, which adjoins the QP subject at a t node of the semantic tree for the VP and substitutes the quantifier's hypothesis in at the object node of the VP. Since there are two sites where adjunction can take place, there are two different semantic trees that can be derived. Thus there are two semantic modes of combination associated with one syntactic rule, which is permitted by Jonnsen compositionality.

The same function is in fact the sole function needed to satisfy Jonnson compositionality, as the reader may check. Even in the case of Qscope ambiguous sentences, whenever a complex VP combines with a DP, this involves adjoining the semantic tree for a quantifier at any t node in the structure and substituting its hypothesis in. There are finitely many ways in which a quantifier can be adjoined to a given tree of finite length, each representing a possibly different implicit semantic rule that corresponds to the syntactic rule that merges the DP subject with the VP.

Let us consider relational nouns, such as "Every representative of some company".

By Jonnsen compositionality, there must be some r such that

(8) $\mu(\sigma(e_D, e_{NP})) = r(\sigma, (\mu e_D, \mu e_{NP}))$

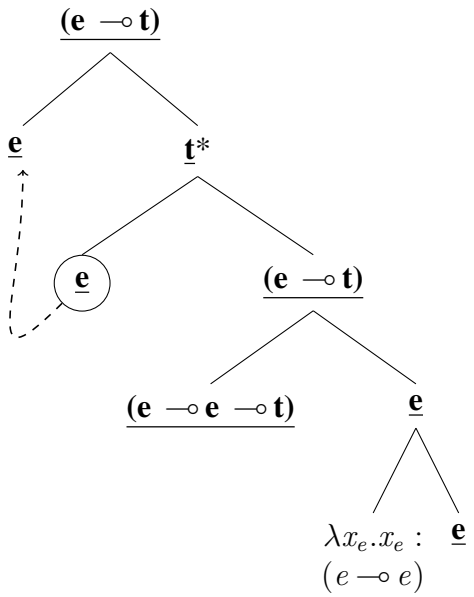
From chapter 2 it will be recalled that a simultaneous operation substitutes the semantic tree for "representative of some company" into the sister of the "Every" node and substitutes a hypothesis into the this tree. So there is an r, namely two joint operations of substitution, that combines μe_D with μe_{NP}. But by Jonnsen compositionality, there must also be r_i, r_j such that

(9) $\mu(\sigma(e_{NP}, e_{PP})) = r_i(\sigma, (\mu e_N, \mu e_{PP}))$

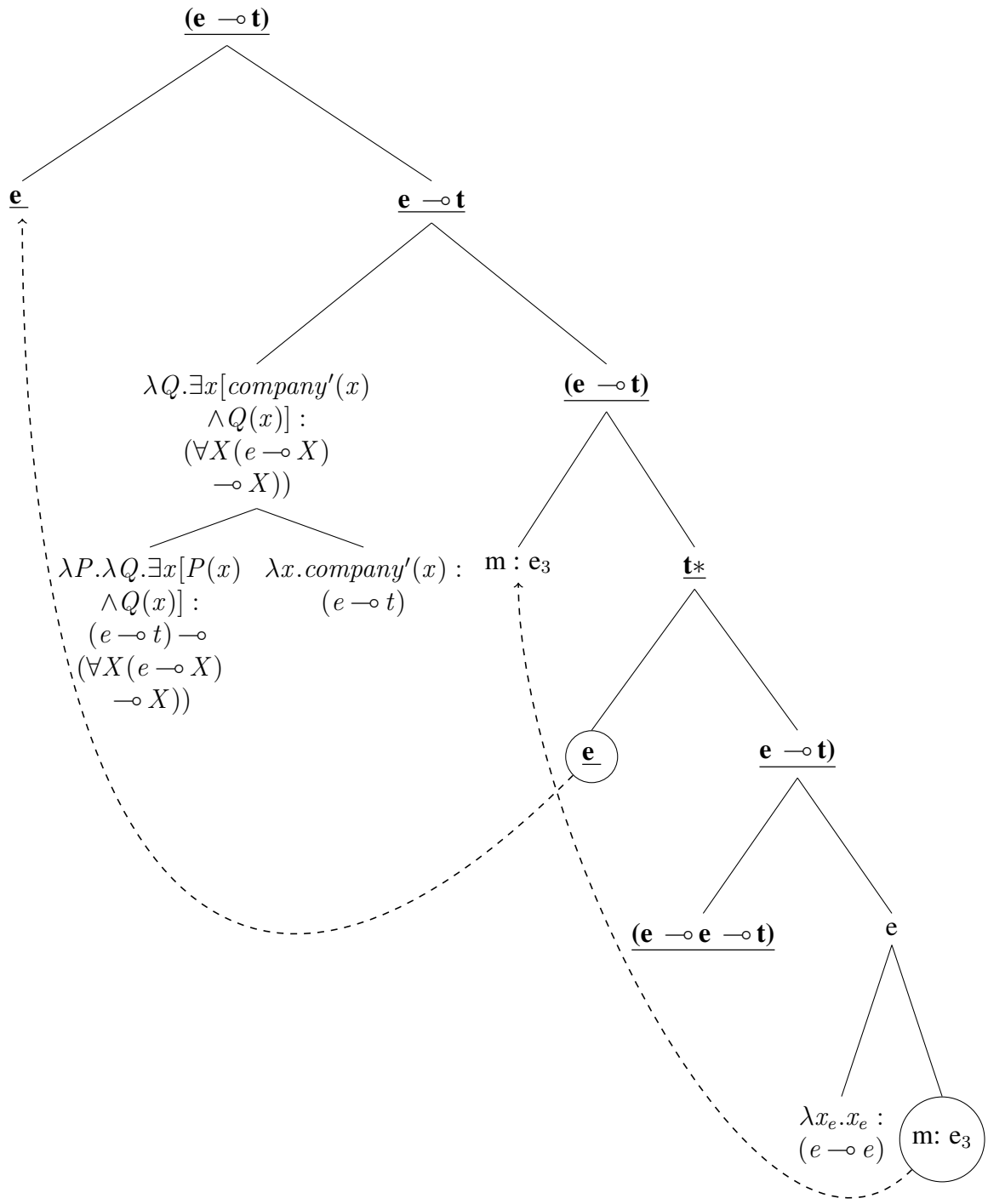
(10) $\mu(\sigma(e_P, e_{DP})) = r_j(\sigma, (\mu e_P, \mu e_{DP}))$

(10) is satisfied by simultaneously adjoining *some company* at the t node of of and substituting in a hypothesis in the bottom-most node of the tree for of. The tree for of and *of some company* are as follows:

(11)

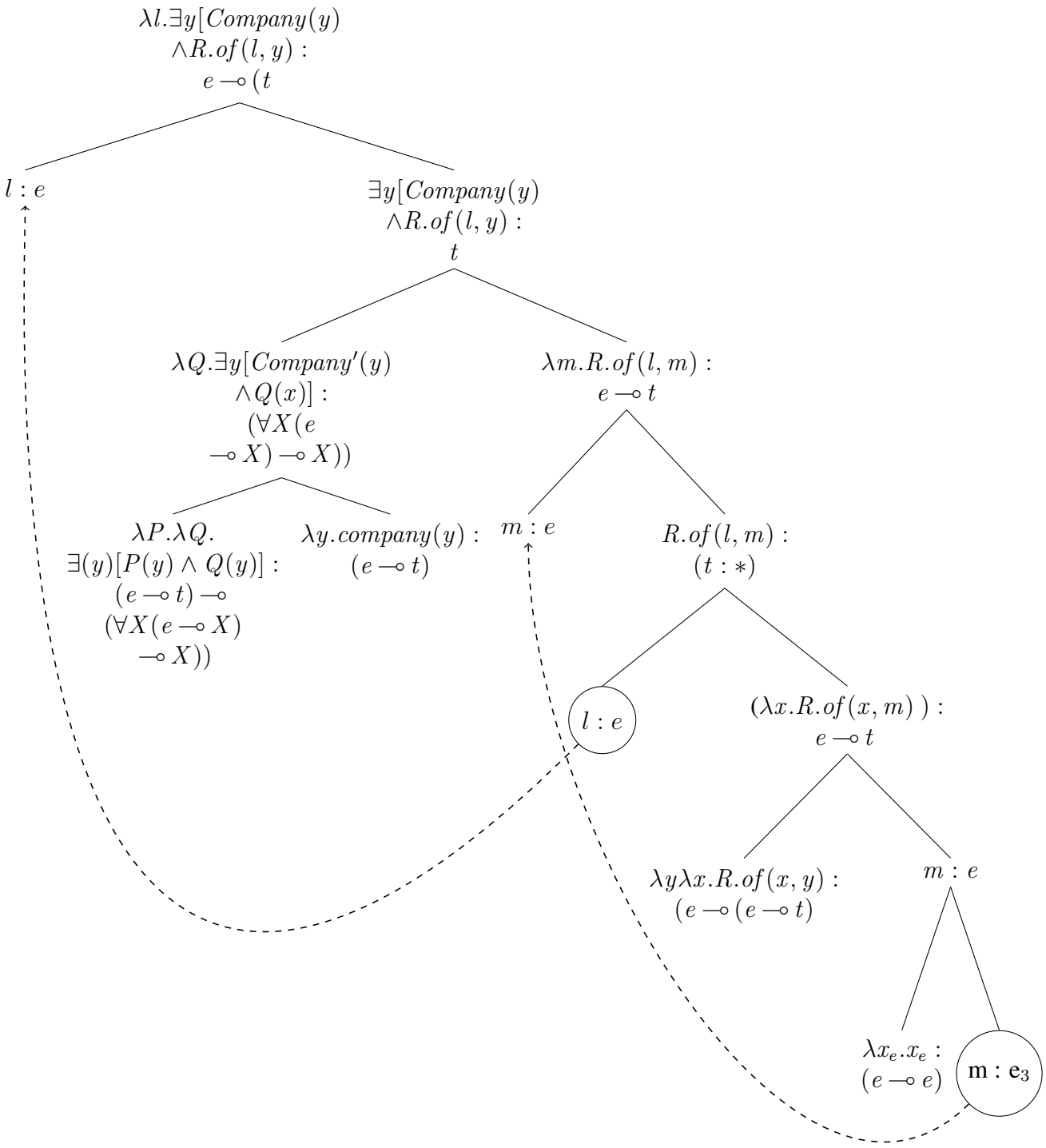


(12)



Finally, we satisfy (9) by a joint operation of substituting the lexical entry for $representative$ at the **(e ⊸ e ⊸ t)** node and substituting its hypothesis l in (it cannot be substituted anywhere else in the tree):

(13)


$\lambda l.\exists y[Company(y) \wedge R.of(l,y) : e \multimap (t$
$l : e$
$\exists y[Company(y) \wedge R.of(l,y) : t$
$\lambda Q.\exists y[Company'(y) \wedge Q(x)] : (\forall X(e \multimap X) \multimap X))$
$\lambda m.R.of(l,m) : e \multimap t$
$\lambda P.\lambda Q. \exists(y)[P(y) \wedge Q(y)] : (e \multimap t) \multimap (\forall X(e \multimap X) \multimap X))$
$\lambda y.company(y) : (e \multimap t)$
$m : e$
$R.of(l,m) : (t : *)$
$l : e$
$(\lambda x.R.of(x,m)) : e \multimap t$
$\lambda y \lambda x.R.of(x,y) : (e \multimap (e \multimap t)$
$m : e$
$\lambda x_e.x_e : (e \multimap e)$
m : e3

An operation then takes this predicate and both simultaneously substitutes it into the semantic tree for $every$ and inserts a hypothesis in the remaining node of the tree. Thus the r referred to in (8) is more complicated than described above.

Note that at this point there is only one place at which "some company" can adjoin, hence there can be no scopal ambiguity. But when "representative of some company" is substituted into the semantic tree for "every", "some company" can adjoin to the root of "every", outscoping it (or it can stay in the original position it has adjoined to). There is a distinct operation of adjunction, $r^{j'}$ for any site at which "some company" can adjoin. This is Jonnsen compositional so long as there is a finite number of implicit semantic rules for a given syntactic rule which adjoin a quantifier in a given place.

3.3.2 Approach 2: Westerståhl's relational compositionality

Westerståhl formulates a version of compositionality on which there is there is a relation between grammatical terms and their semantic values. This gives rise to the following informal statement of relational compositionality:

(14)

(RuleAmb): For every syntactic rule r_{syn} there are corresponding semantic operations $(r_{1,sem}, ..., r_{k,sem})$ such that the meanings of the result of applying r_{syn} to certain expressions is the result of applying $r_{i,sem}$, for some i, to meanings of those expressions.

Take the following examples of Qscope ambiguities, assuming all proper subterms of (16) are unambiguous:

(15) Four critics reviewed three films

(16) $\alpha(\beta(four, critic), \gamma(reviewed, \beta(three, film)))$

On Westerståhl's proposal ambiguity sets in only at the application of the last rule, α, which merges together the VP and the NP (he remarks, p.17 scope) that other accounts are also possible). There are three semantic operators, $r_{\alpha 1}$, $r_{\alpha 2}$, and $r_{\alpha 3}$, corresponding to the syntactic operator α. Interpreting (15) involves selecting the appropriate semantic operator, based on contextual and performance-based factors. This is little different from the sort of contextually-based decision to select an interpretation of a particular lexically ambiguous expression, such as "bank".

The semantic value $\mu_R(p)$ of p is a subset of $E \times M$ (Where E and M are respectively the set of expressions $(p_0, ..., p_i)$ and the set of meanings $(m_0, ..., m_j)$ for i, j):

Definition 24 $\mu(p) = R_p = \{m : R(p, m)\}$

By definition we thus have a set-valued function from a grammatical term to the set of its interpretations.

We may then formalise Westerståhl's relational compositionality as follows. Call a semantics R bounded if for each operation α of the syntax the number of semantic values of a grammatical term of the form $\alpha(q_1, ..., q_n)$ is less than or equal to some finite number k:

Definition 25 $\left| \bigcup_{i=1} \left\{ R_{\alpha(q_1,...,q_n)} : \bigwedge_{1 \leq i \leq n} R(q_i, m_i) \right\} \right| \leq k$

Then a semantics is compositional iff $Rule^k(R)$ for R bounded by k holds:

Westerståhl compositionality
$Rule^k(R)$**:**

For each syntactic rule α there are semantic operations $r^1_\alpha, ..., r^k_\alpha$ such that for each semantic value m of the grammatical term $\alpha(p_1, ..., p_n)$, $m \in R_{\alpha(p_1,...,p_n)}$ there is some j-ary semantic operation, and $m_i \in R_{p_i}$ for each $p_i \in \alpha(p_1, ..., p_n)$, such that

(a) $m = r^j_\alpha(m_1, ..., m_n)$

(b) for any j', $1 \leq j' \leq k, r^{j'}_\alpha(m_1, ..., m_n) \in R_{\alpha(p_1,...,p_n)}$

In the $k = 1$ case $Rule^k(R)$ reduces to Hodges' Funct(μ), the form of compositionality reviewed above.

Clause (a) ensures that for each semantic value m a grammatical term has, there is a unique semantic operation $r^{j'}_\alpha$ that, together with any meanings assigned the immediate constituents, $(p_1, ..., p_n)$ of that grammatical term, produces m. Clause (b) ensures that the semantic value m produced is a subset of $R_{\alpha(p_1,...,p_n)}$, which is to say, it belongs to the following set:

$R_{\alpha(p_1,...,p_n)} = \{m : R(\alpha(p_1, ..., p_n), m)\}$

Let us see if our semantics is compositional in the relational sense isolated by Westerståhl.

First off we have intransitive sentences with proper names as subjects. Let α be the syntactic operation that merges a DP subject (where the DP is either one quantifier or a proper name) with an intransitive VP. So we have that

(17) $\alpha(e_{DP}, e_{VP})$

There is an r^j_α, namely a simultaneous operation on the tree set for the proper name which adjoins the quantifier tree and substitutes its hypothesis in the semantic tree for the verb), such that $\mu(\alpha(e_{DP}, e_{VP}) = r^j_\alpha(\mu(e_{DP}), \mu(e_{VP}))$.

Next we have transitive sentences whose subjects and objects may be either quantifiers or proper names. Let α, β and γ be the relevant syntactic rules:

(18) $\alpha(e_{DP}, e_{VP})$

(19) $\beta(e_V, e_{DP})$

(20) $\gamma(e_D, e_{NP})$

The last two rules we have already accounted for: the last is by substitution of $\mu(e_{NP}$ into $\mu(e_D$ and the penultimate is by the simultaneous operation of adjoining the semantic tree for the quantifier DP to the root of $\mu(e_V$ and substituting its hypothesis in the remaning node of the structure. The first is again by simultaneous adjunction and substitution, so there is an r^j_α such that $\mu(\alpha(e_{DP}, e_{VP}) =$

$r^j_\alpha(\mu(e_{DP}), \mu(e_{VP}))$. But there are two adjunction sites: one at the root of the semantic tree for the VP, and the other at the t node at the root of the semantic tree for the verb. Hence in addition to r^j_α, there is a distinct relation of adjunction $r^{j'}_\alpha$. Each of r^j_α and $r^{j'}_\alpha$ satisfy (a) and (b) where $k = 2$.

What about relational nouns? We have the following syntactic rules:

(21) $(\alpha(e_{NP}, e_{PP})$

(22) $(\beta(e_P, e_{DP})$

Regarding (22), a DP combines with the semantic tree for of by simultaneous adjunction of the DP to the t node and substitution of a hypothesis at the bottom-most node. Regarding (21), NP "representative" combines with the PP "of some company" by substitution of the lexical entry for $representative$ into the semantic tree for of (see the example above) and substitution of its hypothesis in the remaining variable node of the structure. At this point there is only one place at which "some company" can adjoin, hence there can be no scopal ambiguity. But when "representative of some company" is substituted into the semantic tree for "every", "some company" can adjoin to the root of "every", outscoping it (or it can stay in the original position it has adjoined to). There is a distinct operation of adjunction, $r^{j'}$ for any site at which "some company" can adjoin, thus satisfying clause (a) of Westerståhl's definition. The Glue TAG semantics is therefore Westerståhl compositional.

3.4 Conclusion: Comparison with the other accounts of Qscope ambiguity

In this section we will outline a number of similarities and differences between the account of Qscope ambiguity that we have given, and the accounts of Qscope ambiguity that we surveyed in chapter 1. Our conclusion will be that our semantics shares many properties of these semantics. Such similarities may remind the reader of Von Stechow's argument that (i) non-QR based approaches to quantification tend to have a mechanism similar to QR and (ii) in particular, in a Hendriks' style flexible Montague grammar approach "... a closer look at the meta-language in which these type shifters is formulated shows that QR is hidden in the formulation of the rules." However, whereas Von Stechow takes the similarity between QR and other approaches to be problematic for those approaches, for us, this instead suggests a possible convergence at a deeper theoretical level between the different approaches. We will conjecture without conclusive evidence that, at some level of analysis all the approaches end up sharing many formal properties and thus the different methods to Qscope ambiguity should not be seen antagonistically (although they make make different predictions given the principles imposed to restricts quantifier scope).

We will now show how we can reformulate a Cooper-style storage system in our semantics and then look at a concrete example of the similarities between QR and glue. But before, we will briefly consider two broad similarities between glue and other approaches.

One particular similarity between many approaches share is their use of placeholder variables or constants. These allow us to combine a given syntactic construction with a (possibly indexed) pronoun instead of combining it directly with a quantifier. The reader will recall the role of placeholder variables in both Montague's quantifying in and Cooper storage. In both Montague semantics and in Cooper storage, to derive one of the readings of "some boy admires every girl",

"every girl" is substituted with a pronoun, represented as a free variable, that acts as a placeholder for the quantifier (the substitution occurring in the syntax in Montague semantics but in the semantics only in Cooper storage). We then lambda abstract over "some boy admires z_3, to form $\lambda z_3.\exists y.\ boy(y) \wedge\ admires(y, z_3)$, which can then combine with "every girl". Likewise, in Cooper storage, quantifiers can be stored and a variable is used as a placeholder in the computation. When the sentence has been assigned a denotation the rule of retrieval allows us to raise a stored quantifer within the semantics and abstract over the semantic value of the rest of the sentence (see chapter 1). Placeholder variables are also present in QR and in flexible Montague grammar. In QR, traces act essentially as placeholders that allow semantic computation to proceed which are then abstracted over so that the moved quantifer can combine with the structure. In the flexible types approach the placeholder variables are present both in the original denotation of the verb before it is lifted, and nested within the lifted lambda denotation. Like quantifying in, Cooper storage and QR, the TAG glue semantics also uses encircled constants (hypotheses) which may seem to play the role of being placeholders, and which are abstracted over at the point indicated by an arrow. However, in TAG Glue these hypotheses belong to the denotation of a quantifier phrase, which is a tree set containing a quantifier semantic tree and a hypothesis. This enables TAG Glue grammar to provide constants that serve the same purpose as placeholder variables whilst being part of the denotation of quantifiers.

Another broad similarity between the approaches is the fact that they all incorporate something like quantifier movement, whether in semantic or in syntactic form; in Montague grammar and QR movement happens in the syntax, but in Cooper Storage and flexible Montague grammar quantifier movement is effected by operations in the semantics. Our semantics also utilises a form of semantic movement, albeit one that follows from the features of the linear logic we are using.

Glue TAG storage semantics

We now show informally that we can replicate certain features of Cooper semantics in our framework. The two cardinal features of Cooper storage are a mechanism for storing and retrieving quantifier denotations. We replicate cooper storage by enriching the denotations we assign to nodes; a node of the semantic tree is now a tuple of a meaning constructor and a (possibly empty) sequence of stores. In cooper storage variables are substituted with the quantifier denotations, which are placed in the store and paired with an index which refers to the variable that has substituted the quantifier denotation.

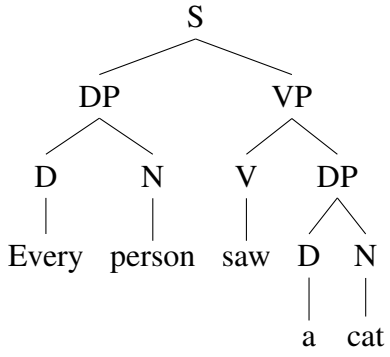
In our reformulation of the storage mechanism in glue, glue proof hypotheses play the role of placeholder variables. The placeholder variables, however play a dual role; they are supplied as arguments to a predicate and placed in the store (where they are bracketed). Retrieval of a variable occurs at the point in the glue proof where a hypothesis is discharged, and this is equivalent to lambda abstraction over the variable. Consequently, the store containing the variable is crossed out and any node higher up in the tree contains a store with the empty set in it, to indicate that the store has been emptied of that particular variable. The one salient difference between Cooper storage and our formulation is that in our formulation the store does not contain a quantifier denotation but a variable, equivalent to the index in the binding store in cooper storage. Thus in our semantics a single item is both introduced as a placeholder in semantic structure and stowed away for further use. The fact that we can represent both of these functions via one mechanism suggests that our approach may be more economic, however, since we have not formalised our reformulation of

Cooper storage our this suggestion is only tentative.

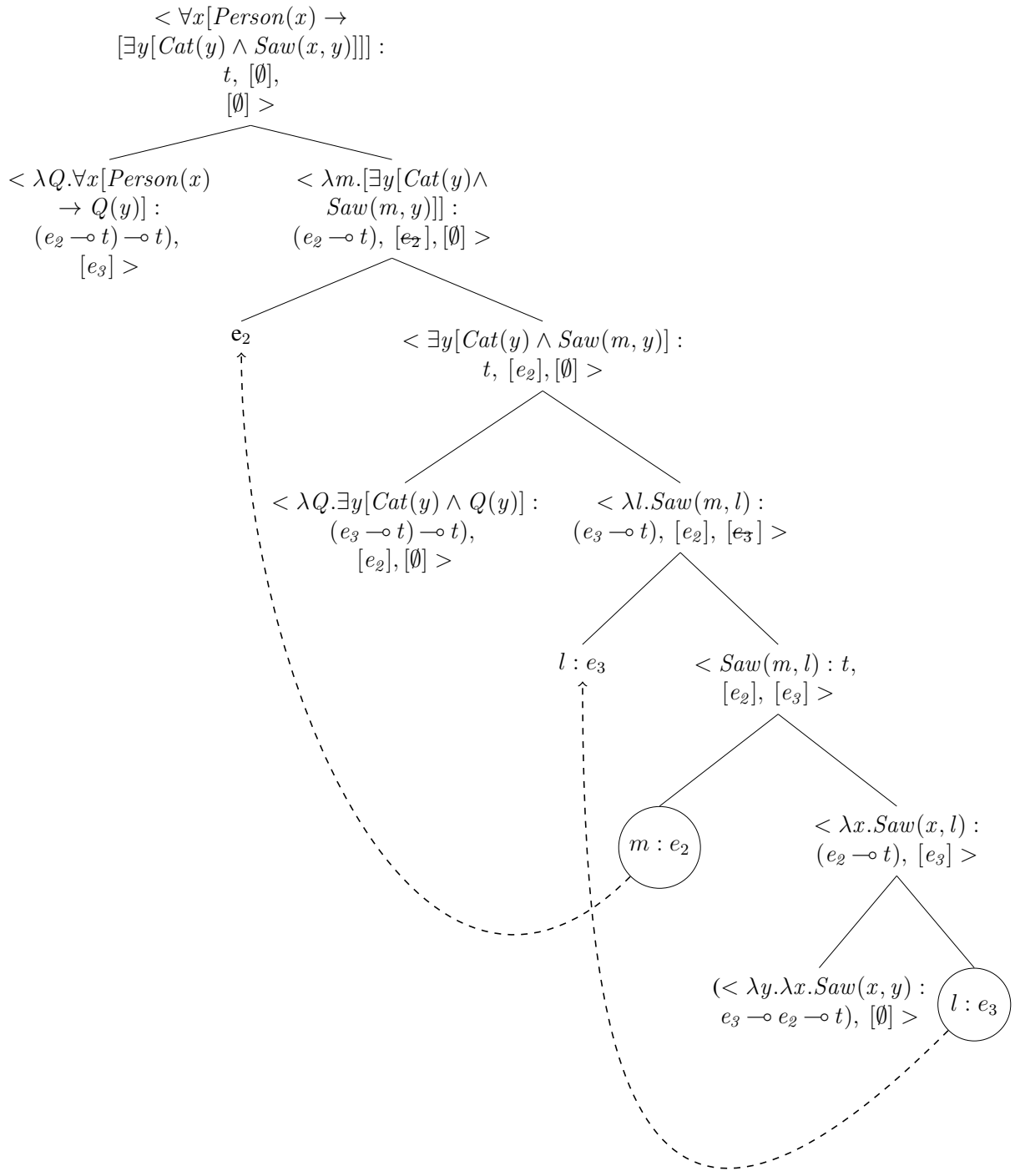
An alternative reformulation of cooper storage within glue might allow the core semantic denotation of a quantifier to be stored alongside an index, as in classical cooper storage. However, the retrieval mechanism would then have to provide a rule whereby the core semantic value of a quantifier and a variable are reintroduced into the non stored part of a semantic tree. Since the mechanism of hypothesis discharging enables this for free, we prefer our reformulation of Cooper storage, although we have not considered fully the form that a more traditional form of cooper storage could take in our semantics.

As an example of our formulation of storage within glue, consider (23) and the following representation of its surface and inverse scope readings, (24):

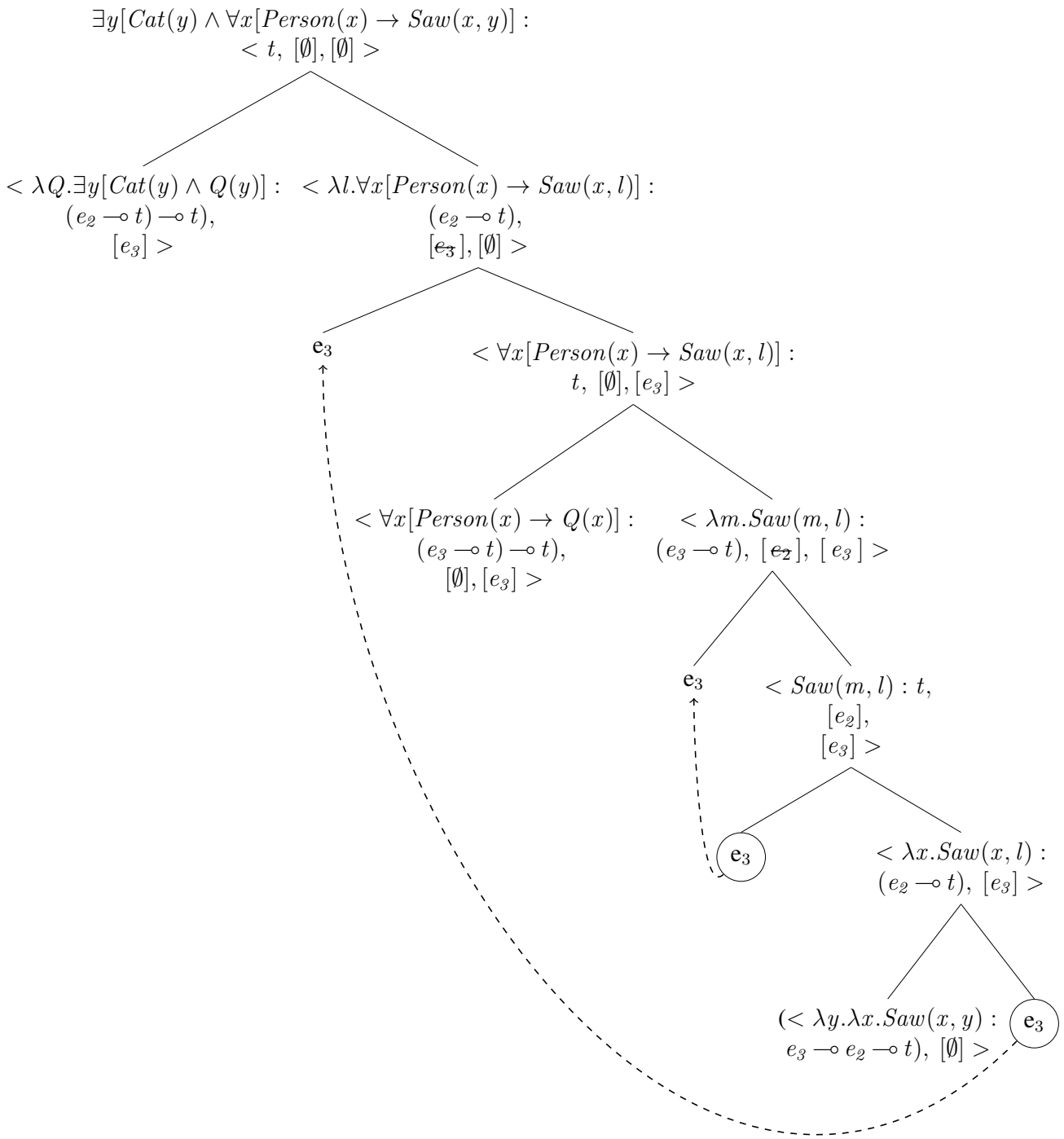
(23)



(24)


$< \forall x[Person(x) \rightarrow [\exists y[Cat(y) \wedge Saw(x,y)]]] : t,\ [\emptyset],\ [\emptyset] >$
$< \lambda Q.\forall x[Person(x) \rightarrow Q(y)] : (e_2 \multimap t) \multimap t),\ [e_3] >$
$< \lambda m.[\exists y[Cat(y) \wedge Saw(m,y)]] : (e_2 \multimap t),\ [\sout{e_2}], [\emptyset] >$
e_2
$< \exists y[Cat(y) \wedge Saw(m,y)] : t,\ [e_2], [\emptyset] >$
$< \lambda Q.\exists y[Cat(y) \wedge Q(y)] : (e_3 \multimap t) \multimap t),\ [e_2], [\emptyset] >$
$< \lambda l.Saw(m,l) : (e_3 \multimap t),\ [e_2],\ [\sout{e_3}] >$
$l : e_3$
$< Saw(m,l) : t,\ [e_2],\ [e_3] >$
$m : e_2$
$< \lambda x.Saw(x,l) : (e_2 \multimap t),\ [e_3] >$
$(< \lambda y.\lambda x.Saw(x,y) : e_3 \multimap e_2 \multimap t),\ [\emptyset] >$
$l : e_3$

(25)



These trees show that we can represent the key features of cooper storage, the introduction of a placeholder and storing a variable away for further use, using glue. We will now briefly compare QR and our semantics, focusing on the role of movement.

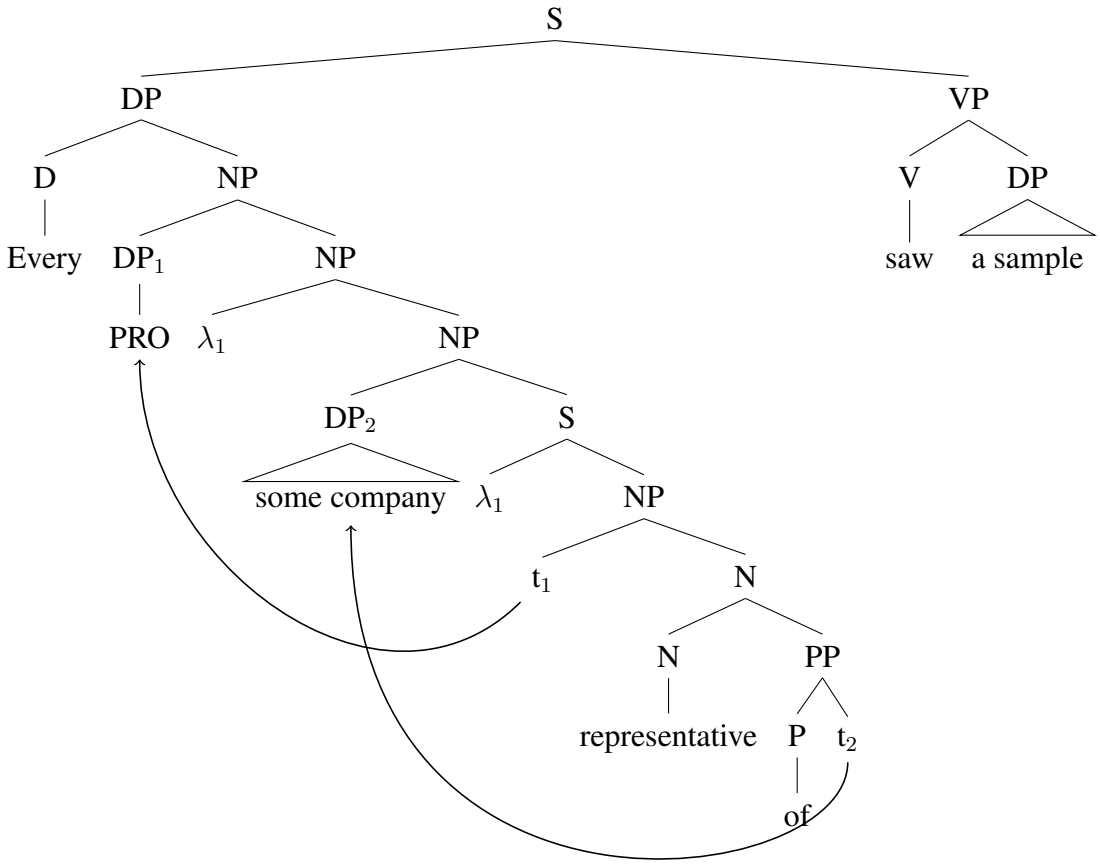
QR and TAG glue

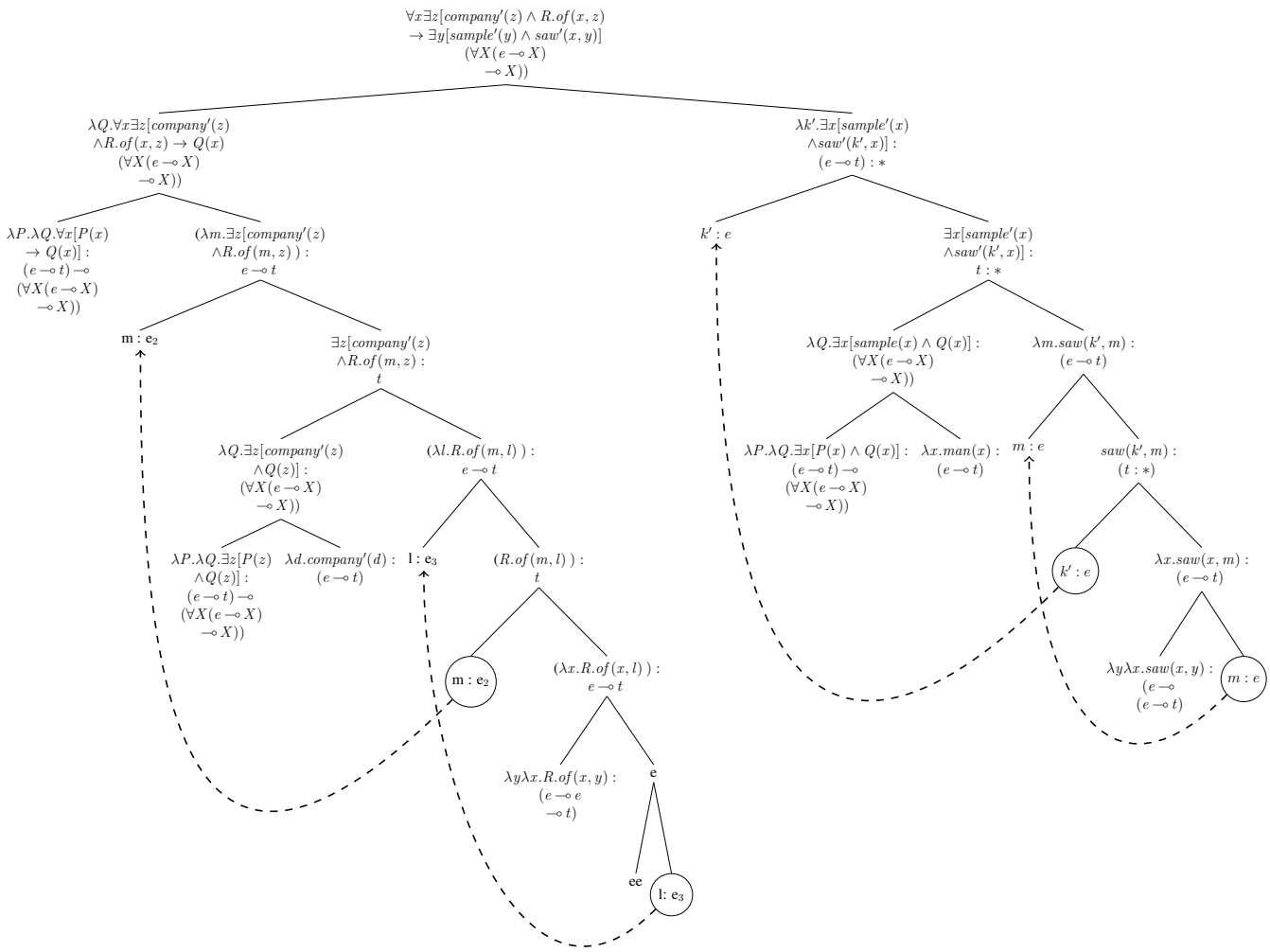
In our semantics, we represent implication introduction in the glue proof as abstraction over a hypothesis that has semantically moved from the place where it satisfies the verbal predicate. Such

movement occurs after a quantifier in the semantic tree simultaneously adjoins at a t node and its associated hypothesis is substituted in. This is similar to QR, which syntactically moves a quantifier phrase from its *in situ* position leaving behind a trace which is abstracted over (via predicate abstraction) and combined with the denotation of the moved quantifier phrase. However, in TAG glue the hypothesis is part of the dentation of the quantifier.

The main difference between QR and the form of movement in glue is that predicate abstraction is a stipulated rule of composition, that doesn't follow strictly from any of the other rules of composition that Heim and Kratzer employ, whereas abstraction in glue proof semantic trees simply follows from the nature of the logic used. Since the rule governing hypothesis introduction and discharge in glue proofs accounts for both abstraction and for the interpretation of the hypothesis, whereas both predicate abstraction and the traces and pronoun rule are required in the Heim and Kratzer system, our semantics is more economical on this front. However, since we introduce both a core and a peripheral semantic value for expressions it may be thought that this economy comes at a certain price (however we motivate this distinction in chapter 2).

The similarities between QR and the TAG Glue semantics that we have discussed can be seen by considering the following LF and semantic tree of "Every representative of a company saw a sample":





As is apparent, both these trees share many features in common (viz. those mentioned above). The main difference as far as we are concerned is that predicate abstraction in the LF above is a stipulated rule of composition, that doesn't follow from any of the general features of the set up that Heim and Kratzer employ. On the other hand, abstraction in glue proof semantic trees simply follows from the nature of the logic used: it is for free. Moreover, as we explained in the preceeding chapter, in our semantics the abstraction site is part of the semantics of quantifiers (part of the perhiperhal semantic value of a quantifier). Thus what we gain with the glue proof semantic trees is a reduction in certain composition rules, albeit we require two sorts of semantic value (core semantic value and peripheral semantic value (which we argued were motivated by various considerations).

3.5 Conclusion

We will then be able to consider how they combine, and what mechanisms are involved:

Bibliography

Adger, D. 2003. *Core syntax: A minimalist approach*, volume 33. Oxford University Press Oxford.

Aoun, J. and Li, Y.-H. A. 1993. *Syntax of scope*, volume 21. Mit Press.

Asudeh, A. 2012. *The logic of pronominal resumption*. Oxford University Press.

Asudeh, A. and Crouch, R. 2001a. Glue semantics: A general theory of meaning composition. Talk given at Stanford Semantics Fest 2, March 16.

Asudeh, A. and Crouch, R. 2001b. Glue semantics for hpsg. *Technology* 3.

Asudeh, A., Crouch, R., Butt, M., and King, T. H. 2002. Coordination and parallelism in glue semantics: Integrating discourse cohesion and the element constraint. In *Proceedings of the LFG02 Conference*, 19–39, Citeseer.

Bach, E. 1976. An extension of classical transformational grammar. *in Problems in Linguistic Metatheory (Proceedings of the 1976 conference), pp. 183224. East Lansing, Michigan: Michigan State University* .

Barker, C. 2002. Continuations and the nature of quantification. *Natural language semantics* 10:211–242.

Barker, C. and Jacobson, P. I. 2007. Direct compositionality .

Barker, C. and Shan, Chung-chieh. 2014. *Continuations and natural language*, volume 53. Oxford University Press.

Bernardi, R. 2010. Scope ambiguities through the mirror. *The Linguistics Enterprise: Cognition, Methodology, Modelling* 11–54.

Blackburn, P. and Bos, J. 2005. Representation and inference for natural language. *A first course in computational semantics. CSLI* .

Blackburn, P. and Meyer-Viol, W. 1997. Modal logic and model-theoretic syntax. In *Advances in Intensional Logic*, 29–60, Springer.

Carnie, A. 2010. *Constituent structure*, volume 5. Oxford University Press.

Carpenter, B. 1997. *Type-logical semantics*. MIT press.

Chiswell, I. and Hodges, W. 2007. *Mathematical logic*, volume 3. Oxford University Press.

Chomsky, N. 1965. *Aspects of the Theory of Syntax*. 11, MIT press.

Chomsky, N. 1986. *Knowledge of language: Its nature, origin, and use*. Greenwood Publishing Group.

Cooper, R. (1983). *Quantification and Syntactic Theory*. Dordrecht: Reidel.

Cresswell, M. J. 1973. Logics and languages .

Cresswell, M. J. 1982. The autonomy of semantics. In *Processes, Beliefs, and Questions*, 69–86, Springer.

Crouch, R. and van Genabith, J. 2000. Linear logic for linguists. *URL: http://www2. parc. com/istl/members/crouch* .

Dalrymple, M. 1999. *Semantics and syntax in Lexical Functional Grammar: The resource logic approach.*. Cambridge, MA: MIT Press.

Di Cosmo, R. and Miller, D. 2006. Linear logic .

Dowty, D. 1979. Word meaning and montague grammar: The semantics of verbs and times in generative semantics and in montague's ptq, d. *Reidel, Dordrecht* .

Dowty, D. R., Wall, R., and Peters, S. 2012. *Introduction to Montague semantics*, volume 11. Springer Science & Business Media.

Dummett, M. 1973. *Frege: Philosophy of Language*. London: Gerald Duckworth.

Dummett, M. 1993. What do i know when i know a language? .

Evans, G. 1981. Semantic theory and tacit knowledge .

Fox, C. and Lappin, S. 2008. *Foundations of intensional semantics*. John Wiley & Sons.

Frank, R. 2004. *Phrase structure composition and syntactic dependencies*, volume 38. MIT Press.

Gallin, D. 1975. Intensional and higher-order modal logic: with applications to montague semantics .

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science* .

Glanzberg, M. 2009. Semantics and truth relative to a world. *Synthese* 166:281–307.

Gotham, M. 2015. Towards glue semantics for minimalist syntax. *Cambridge Occasional Papers in Linguistics* 8:5683.

Graf, T. 2013. Local and transderivational constraints in syntax and semantics. Ph.D. thesis, University of California Los Angeles.

Gratzer, G. A. 2008. *Universal algebra*. Springer Science & Business Media.

Han, Chung-Hye and Hedberg, N. 2008. Syntax and semantics of it-clefts: a tree adjoining grammar analysis. *Journal of Semantics* 25:345–380.

Heim, I. and Kratzer, A. 1998. *Semantics in generative grammar*, volume 13. Blackwell Oxford.

Hendriks, H. 2001. Compositionality and model-theoretic interpretation. *Journal of Logic, Language and Information* 10:29–48.

Hendriks, H. L. W. 1993. *Studied flexibility: Categories and types in syntax and semantics*. Institute for Logic, Language and Computation.

Henkin, L. 1950. Completeness in the theory of types. *The Journal of Symbolic Logic* 15:81–91.

Hodges, W. 2001. Formal features of compositionality. *Journal of Logic, Language and Information* 10:7–28.

Hodges, W. 2004. Compositionality is not the problem. *Logic and Logical Philosophy* 6:7–33.

Hodges, W. 2012. Requirements on a theory of sentence and word meanings. *Prospects for Meaning. Berlin* 583–608.

Jacobson, P. 2002. The (dis) organization of the grammar: 25 years. *Linguistics and Philosophy* 25:601–626.

Janssen, T. 1983. Foundations and applications of montague grammars. Ph.D. thesis, University of Amsterdam.

Janssen, T. 1997. Compositionality .

Jönsson, M. 2008. *On Compositionality*. Lund University.

Joshi, A. K., Kallmeyer, L., and Romero, M. 2007. Flexible composition in ltag: Quantifier scope and inverse linking. In *Computing meaning*, 233–256, Springer.

Joshi, A. K. and Schabes, Y. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, 69–123, Springer.

Katz, J. J. 1980. *Language and other abstract objects*.

Keenan, E. L. and Stabler, E. P. 1999. *Bare grammar*. Citeseer.

Keller, W. R. 1988. *Nested cooper storage: The proper treatment of quantification in ordinary noun phrases*. Springer.

Kelsey, R., Clinger, W., and Rees, J. 1998. The revised5 report on the algorithmic language scheme–higher-order and symbolic computation, 11 (1).

Klein, E. and Sag, I. A. 1985. Type-driven translation. *Linguistics and Philosophy* 8:163–201.

Kornai, A. and Pullum, G. K. 1990. The x-bar theory of phrase structure. *Language* 24–50.

Lewis, D. 1970. General semantics. *Synthese* 22:18–67.

Lewis, D. 1981. Index, context, and content. In *Philosophy and grammar*, 79–100, Springer.

Lewis, D. 2008. *Convention: A philosophical study*. John Wiley & Sons.

Martin-Löf, P. 1996. On the meanings of the logical constants and the justifications of the logical laws. *Nordic journal of philosophical logic* 1:11–60.

Montague, R. 1975. Formal philosophy .

Murzi, J. and Steinberger, F. 2015. Inferential role semantics. *in preparation for A Companion to the Philosophy of Language (Second Edition), B. Hale, A. Miller and C. Wright (eds), Blackwell* .

Ninan, D. 2010. Semantics and the objects of assertion. *Linguistics and Philosophy* 33:355–380.

Pagin, P. 2003. Communication and strong compositionality. *Journal of Philosophical Logic* 32:287–322.

Pagin, P. 2012. Communication and the complexity of semantics .

Pagin, P. and Westerståhl, D. 2010. Compositionality i: Definitions and variants. *Philosophy Compass* 5:250–264.

Partee, B. 1975. Montague grammar and transformational grammar. *Linguistic inquiry* 203–300.

Partee, B., Ter Meulen, A., and Wall, R. 2012. *Mathematical methods in linguistics*, volume 30. Springer Science & Business Media.

Pelletier, F.J. 1999. Semantic compositionality: Free algebras and the argument from ambiguity. .

Peters, S. and Westerståhl, D. 2006. Quantifiers in language and logic .

Quine, W. V. 1972. *Methodological reflections on current linguistic theory*. Springer.

Rabern, B. 2012. Against the identification of assertoric content with compositional value. *Synthese* 189:75–96.

Restall, G. 2002. *An introduction to substructural logics*. Routledge.

Restall, G. 2006. Relevant and substructural logics. *Handbook of the History of Logic* 7:289–398.

Ruys, E. G. and Winter, Y. 2011. Quantifier scope in formal linguistics. In *Handbook of philosophical logic*, 159–225, Springer.

Scholz, F., B. Pelletier and Pullum, G. 2015. Philosophy of linguistics .

Stabler, E. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, 68–95, Springer.

Stabler, E. P. 2011. Meta meta linguistics. *Theoretical Linguistics* 37:69–78.

Stanley, J. 1997. Names and rigid designation. *A Companion to the Philosophy of Language* 555–85.

Steedman, M. 2012. *Taking scope: The natural semantics of quantifiers*. MIT Press.

Stokhof, M. and Van Lambalgen, M. 2011. Abstractions and idealisations: The construction of modern linguistics. *Theoretical linguistics* 37:1–26.

Thomason, R. H. 1980. A model theory for propositional attitudes. *Linguistics and Philosophy* 4:47–70.

Von Stechow, A. 1990. Categorial grammar and linguistic theory. *Studies in Language* 14:433478.
Von Stechow, A. 2009. Syntax and semantics: an overview. *Manuscript, University of Tübingen* .
Weir, D. 1988. Characterizing mildly context-sensitive grammar formalisms. Ph.D. thesis, University of Pennsylvania.
Werning, M., Hinzen, W., and Machery, E. 2012. *The Oxford handbook of compositionality*. Oxford University Press.
Westerstahl, D. 2007. Compositionality and ambiguity .
Williams, E. 1983. Semantic vs. syntactic categories. *Linguistics and philosophy* 6:423–446.
Williams, J. R. G. 2005. The inscrutability of reference. Ph.D. thesis, University of St. Andrews.
Yalcin, S. 2014. Semantics and metasemantics in the context of generative grammar. *Metasemantics: New Essays on the Foundations of Meaning* 17.
Zimmermann, T. E. 1999. Meaning postulates and the model-theoretic approach to natural language semantics. *Linguistics and Philosophy* 22:529–561.
Zimmermann, T. E. 2006. The values of semantics. In *Form, Structure, and Grammar: A Festschrift Presented to Günther Grewendorf on Occasion of His 60th Birthday*, 383, Walter de Gruyter GmbH & Co KG.
Zimmermann, T. E. 2011. Model-theoretic semantics. *Semantics: An International Handbook of Natural Language Meaning* 1:762–802.
Zimmermann, T. E. 2012. Equivalence of semantic theories. *Prospects for Meaning. Berlin* 629–649.