

# Categorial Grammar with Ontology-refined Types

Gilad Ben-Avi<sup>a</sup>, Nissim Francez<sup>a</sup>

<sup>a</sup>*Department of Computer Science, The Technion, Haifa, Israel*

---

## Abstract

In this paper we present a modified version of the associative Lambek calculus, in which semantically bad, but grammatically well formed, sentences like **every table sang** are not derived. For this purpose we use ontology  $\mathcal{O}_\sigma$  that structures the domain of interpretation of a type  $(\sigma, t)$ . The concepts in  $\mathcal{O}_\sigma$  are used to refine a type  $\sigma$ , in such a way that the selectional restrictions that a function with argument of type  $\sigma$  imposes on this argument can be defined in a more subtle manner than in the standard Lambek calculus.

*Key words:* Categorial Grammar, Lambek Calculus, Selectional Restrictions, Ontologies.

---

## 1 Introduction

In Type-Logical grammars (TLG) ([1]), under their usual type-driven syntax-semantics interface, sentences (1a) and (1b) below have isomorphic derivations, because (under a standard lexicon) both the syntactic category and the semantic type of **boy** and **table** are the same.

- (1)a. Every boy smiled.  
b. ? Every table smiled.

Similarly, all the expressions in (2) are well-formed verb phrases, and all the expressions in (3) are well-formed nouns.

- (2)a. saw every table

---

*Email addresses:* bagilad@cs.technion.ac.il (Gilad Ben-Avi), francez@cs.technion.ac.il (Nissim Francez).

- b. ? met every table
- (3)a. young lion
- b. young boy
- c. ? young table

Furthermore, under standard model-theoretic semantics, even the *denotations* of **boy** and **table** cannot be controlled, being each an *arbitrary* set of objects in a model.

Thus, in spite of its mathematical elegance, the above mentioned observations are considered a weakness of the approach, which has to be overcome. This has been recognized ever since Montague semantics was proposed in e.g., [2,3], where *meaning postulates* were proposed as a remedy, restricting models and denotations in them. However, resorting to such means complicates the semantics significantly.

This paper attempts to extend TLG with a simpler disciplined means, compatible with the type drivenness of the grammar, by means of which *selectional restrictions* can be imposed to eliminate expressions like those in (1b), (2b) and (3c), which are well-formed syntactically, but ill-formed semantically.

A tool frequently used in *lexical semantics* is *ontologies* (see for example [4–6]), which structure domains of interpretation via a partial order of *specificity*. Such ontologies can express many meaning postulates without an explicit logical statement of them. For example, the denotations of **boy** and **table** can be made disjoint by structuring the domain of type  $e$ .

The main idea is to use an ontology for a type  $\sigma$  to *refine*  $\sigma$  in a way that enables the restriction of functions that standardly apply to *any* argument of type  $\sigma$  to apply only to arguments that correspond to some concepts in the ontology for  $\sigma$ . For example, assume that in an ontology for type  $e$  there are disjoint concepts *animate* and *inanimate*. Furthermore, assume it is possible to associate with **boy** a concept that is at least as specific as *animate*. Similarly, a concept can be associated with **table** that is at least as specific as *inanimate*. Now, the denotation of the verb **smile** (a function of type  $(e, t)$ ) will be restricted to apply to arguments of type  $e$  that are *animate* (or more specific). Since the denotation of **boy** is such, but not the denotation of **table**, sentence (1a) will be derivable in our system and sentence (1b) will not. A similar story applies to the interaction between the transitive verb and its object in (2a) and (2b), and to the adjectival modification in (3a), (3b) and (3c).

This refinement of types by ontologies is embedded within TLG (here, grammars that are based on the associative Lambek calculus ([7]), henceforth **L**) by incorporating the ontological concepts as *decorations* of types, and extending **L** to respect such refined types, that originate from the lexicon. The formalism

also uses variables ranging over concepts, to be explained below.

The idea of augmenting a logic with an ontology was suggested before in Knowledge Representation and Logic Programming (e.g., [8,9]). However, there the sorted information is used during *unification* in refuting a goal, a totally different use. Another connection is to  $\lambda$ -calculus with subtyping (e.g., via intersection types [8]). There also, the main issue is higher-order unification terms of such types. Our use of concepts to restrict functional application, and of variables ranging over concepts participating in type refinements, seem to be novel.

## 2 Ontologies

Consider a standard definition of semantic types for TLG. We augment TLGs so that to any semantic type  $\sigma$  there correspond an ontology  $\mathcal{O}_\sigma$ , that enforces some restrictions on the domain of interpretation  $D_{(\sigma,t)}$  of  $(\sigma, t)$  in every model. Formally,  $\mathcal{O}_\sigma$  is the join semi-lattice,  $\langle C_{\mathcal{O}_\sigma}, \leq_{\mathcal{O}_\sigma} \rangle$ . Here  $C_{\mathcal{O}_\sigma}$  is a set of *concepts* with a top element,  $\top_\sigma \in C_{\mathcal{O}_\sigma}$ . The trivial ontology on  $\sigma$  is  $\mathcal{O}_\sigma = \{\top_\sigma\}$ . The partial order  $\leq_{\mathcal{O}_\sigma}$  is called *concept hierarchy*. We usually omit the subscript  $\mathcal{O}$ , whenever this does not give rise to confusion. Additionally, let  $V_{\mathcal{O}_\sigma}$  be a countable set of *ontological variables* ranging over concepts in  $C_{\mathcal{O}_\sigma}$ . We use  $X, Y, Z$ , possibly subscripted, as such variables. We assume that for  $\sigma_1 \neq \sigma_2$  both  $C_{\mathcal{O}_{\sigma_1}} \cap C_{\mathcal{O}_{\sigma_2}} = \emptyset$  and  $V_{\mathcal{O}_{\sigma_1}} \cap V_{\mathcal{O}_{\sigma_2}} = \emptyset$ .

Concepts have denotations in models of the types. For a concept  $c \in C_{\mathcal{O}_\sigma}$ ,  $\llbracket c \rrbracket^M \subseteq D_{(\sigma,t)}$ . Intuitively, this is the set of all and only those objects of type  $\sigma$  that fall under the concept  $c$ . Only two restrictions are imposed on these denotations. The first constraint is that, for any two concepts  $c_1$  and  $c_2$ , if  $c_1 \leq_{\mathcal{O}_\sigma} c_2$  then  $\llbracket c_1 \rrbracket^M \subseteq \llbracket c_2 \rrbracket^M$  must hold in every model  $M$ . In particular, we assume that  $\llbracket \top_\sigma \rrbracket = D_{(\sigma,t)}$ . Note that, as a consequence,  $\llbracket c_1 \rrbracket \cup \llbracket c_2 \rrbracket \subseteq \llbracket c_1 \vee c_2 \rrbracket$ . The second constraint is that if  $c_1$  and  $c_2$  are *incomparable*, then  $\llbracket c_1 \rrbracket^M \cap \llbracket c_2 \rrbracket^M = \emptyset$ .

In the examples we use the simple ontology for type  $e$ , given in figure 1.<sup>1</sup> The concepts are represented by circled nodes. For two concepts  $c_1$  and  $c_2$ ,  $c_1 \leq c_2$ , i.e.,  $c_1$  is more specific than  $c_2$ , iff the two nodes are connected, and  $c_1$  is below  $c_2$ . For example, *rigid*  $\leq$  *material*. Accordingly, in any model  $M$ ,  $\llbracket rigid \rrbracket^M \subseteq \llbracket material \rrbracket^M$  must hold.

<sup>1</sup> Of course, this ontology is very simple, and is used here only as a running example. Questions about the ontological model, such as what concepts should be represented in an ontology, are not within the scope of this paper.

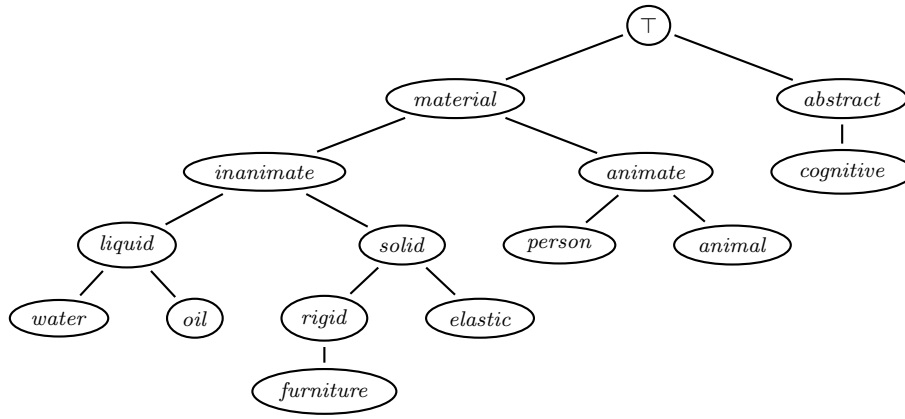


Fig. 1. An example of ontology for type  $e$ .

Table 1

A lexicon

Word	Category	Type	$\lambda$ -term
every	$((s \leftarrow (np \rightarrow s)) \leftarrow n)$ $((s \leftarrow np) \rightarrow s) \leftarrow n$	$((e^X, t), ((e^Y[\geq X], t), t))$	$\lambda A^{(e^X, t)} \lambda B^{(e^Y[\geq X], t)} . \forall x^{e^X} . A(x) \rightarrow B(x)$
some	$((s \leftarrow (np \rightarrow s)) \leftarrow n)$ $((s \leftarrow np) \rightarrow s) \leftarrow n$	$((e^X, t), ((e^Y[\geq X], t), t))$	$\lambda A^{(e^X, t)} \lambda B^{(e^Y[\geq X], t)} . \exists x^{e^X} . A(x) \wedge B(x)$
smile	$(np \rightarrow s)$	$(e^{anm}, t)$	$\lambda x^{e^{anm}} . \mathbf{smile}^{(e^{anm}, t)}(x)$
see	$((np \rightarrow s) \leftarrow np)$	$(e^\top, (e^{anm}, t))$	$\lambda x^{e^\top} \lambda y^{e^{anm}} . \mathbf{see}^{(e^\top, (e^{anm}, t))}(x)(y)$
meet	$((np \rightarrow s) \leftarrow np)$	$(e^{anm}, (e^{anm}, t))$	$\lambda x^{e^{anm}} \lambda y^{e^{anm}} . \mathbf{meet}^{(e^{anm}, (e^{anm}, t))}(x)(y)$
boy	$n$	$(e^{per}, t)$	$\lambda x^{e^{per}} . \mathbf{boy}^{(e^{per}, t)}(x)$
lion	$n$	$(e^{animal}, t)$	$\lambda x^{e^{animal}} . \mathbf{lion}^{(e^{animal}, t)}(x)$
table	$n$	$(e^{fur}, t)$	$\lambda x^{e^{fur}} . \mathbf{table}^{(e^{fur}, t)}(x)$
idea	$n$	$(e^{cog}, t)$	$\lambda x^{e^{cog}} . \mathbf{idea}^{(e^{cog}, t)}(x)$
red	$(n \leftarrow n)$	$((e^X[\leq mat], t), (e^X, t))$	$\lambda x^{(e^X[\leq mat], t)} . \mathbf{red}^{((e^X[\leq mat], t), (e^X, t))}(x)$
young	$(n \leftarrow n)$	$((e^X[\leq anm], t), (e^X, t))$	$\lambda x^{(e^X[\leq anm], t)} . \mathbf{young}^{((e^X[\leq anm], t), (e^X, t))}(x)$

### 3 Semantic types and syntactic categories

An entry in the lexicon in table 1 assigns to each word a syntactic category<sup>2</sup>, a semantic type and a  $\lambda$ -term<sup>3</sup> (its meaning). Note that in some of the superscripts we use abbreviations of the concept names from the ontology in Figure 1: *anm*, *per*, *fur*, *cog* and *mat* abbreviate *animate*, *person*, *furniture*, *cognitive* and *material*, respectively. Except for the superscripts in types, everything in this lexicon is standard. In particular, by ignoring the superscript, both the good and the semantically bad expressions in (1)-(3) can be derived

<sup>2</sup> We use an arrow notation for complex categories that differs from the standard slash/backslash notation. We believe that this notation represents the directionality of a complex category more intuitively.

<sup>3</sup> We use a convention in which types are written only once for a variable or constant in a term; for bound variables, the type will decorate their binding occurrence.

from this lexicon in a standard **L**-grammar. To resolve this and similar difficulties, we refine some of the types by decorating them systematically with concept names, or with variables ranging over concept names. In a preliminary discussion we explain briefly and informally how this is done. The formal details follow.

The type  $(e, t)$  of a common noun is decorated with the most specific appropriate concept  $c \in \mathcal{O}_e$  for the noun in the following way:  $(e^c, t)$ . This decoration indicates that in any model  $M$  the denotation of the noun must be a *subset* of  $\llbracket c \rrbracket^M$ . For example, the noun **boy** is of type  $(e^{person}, t)$ , while the noun **table** is of type  $(e^{furniture}, t)$ . As for verbs, each  $e$  argument in a verb's type  $((e, t)$  for intransitive verbs,  $(e, (e, t))$  for transitive verbs, etc.) is also decorated with a concept  $c \in \mathcal{C}_{\mathcal{O}_e}$ , which is the most general concept to which a respective argument may belong. For instance, intransitive and transitive verbs have types of the form  $(e^c, t)$  and  $(e^{c_1}, (e^{c_2}, t))$ , respectively. These decorations indicate that an argument  $x$  of e.g., the intransitive verb must be a *member* of  $\llbracket c \rrbracket^M$  in any model  $M$ . Note that if  $x \in \llbracket c' \rrbracket^M$  for  $c' \leq c$ , then also  $x \in \llbracket c \rrbracket^M$ . This is due to the requirement that  $\llbracket c' \rrbracket^M \subseteq \llbracket c \rrbracket^M$ . For example, imposing that the subject of **smile** be an animate object is by assigning the type  $(e^{animate}, t)$  to **smile**.

The connection between the noun's concept and the concept licensed by the verb is reflected in the determiner's refined type. The type of a determiner like **every** is also decorated. This time, however, we make use of variables ranging over concepts. The type of **every**, for example, is  $((e^X, t), ((e^{Y[\geq X]}, t), t))$ . If the denotation of **every** applies to the denotation of the noun **boy**, then  $(e^X, t)$  is *matched* against the type  $(e^{person}, t)$  of **boy**. By the definition of type matching (definition 4) this matching succeeds. Furthermore, in the resulting type all other occurrences of  $X$  are instantiated to the value *person*. Thus, the type of the noun phrase **every boy** is set to be  $((e^{Y[\geq person]}, t), t)$ . The decoration  $Y[\geq person]$  indicates that an expression of this type can apply to any expression of type  $(e^c, t)$ , provided  $c \geq person$ . When the denotation of the noun phrase applies to the denotation of a verb like **smile**,  $(e^{Y[\geq person]}, t)$  is matched against the type  $(e^{animate}, t)$  of **smile**. The application succeeds because  $animate \geq person$ . However, if the same determiner meaning applies to the meaning of **table**, whose semantic type is  $(e^{furniture}, t)$ , the resulting noun phrase **every table**, again by matching  $X$ , is of type  $((e^{Y[\geq furniture]}, t), t)$ . In this case,  $animate \not\geq furniture$ , avoiding a derivation of sentence (1b).

The lexicon in table 1 includes (restrictive) adjectives as well. Consider, for example, the adjective **young**. Its decorated type imposes that **young** can modify any noun, provided the noun's denotation falls under *animate*. Also, the resulting type of the modified noun must agree in concept with that of the original noun. Thus, assuming that the types of the words in (3a-c) are like in

table 1, the type of the expression in (3a) is  $(e^{animal}, t)$ , that of the expression in (3b) is  $(e^{person}, t)$ , while (3c) is ill-formed. We assign an adjective a type  $((e^{X[\leq c]}, t), (e^X, t))$ . As in noun phrases, the decoration  $X[\leq c]$  indicates that an expression of this type can apply to any expression of type  $(e^{c'}, t)$ , provided  $c' \leq c$ . The type of the resulting expression will be  $(e^{c'}, t)$  as well ( $X$  is instantiated to  $c'$ ).

We now turn to the formal details. Let  $\mathcal{OT}$  be a finite set of *basic ordered semantic types*, and let  $\mathcal{SB}$  be a finite set of *basic semantic types*, such that  $\mathcal{OT} \subseteq \mathcal{SB}$ . In what follows, we assume for simplicity that  $\mathcal{OT} = \{t\}$ . For every ontology  $\mathcal{O}_\sigma$ , we define the set of all possible decorations of the type  $\sigma$  as  $\text{Dec}_{\mathcal{O}_\sigma} = \{c : c \in C_{\mathcal{O}_\sigma}\} \cup \{X[\geq x] : X \in V_{\mathcal{O}_\sigma}, x \in C_{\mathcal{O}_\sigma}\} \cup \{X[\leq x] : X \in V_{\mathcal{O}_\sigma}, x \in C_{\mathcal{O}_\sigma}\}$ . The set  $\mathcal{ORT}$  of refined types, together with a mapping to the respective underlying undecorated types (denoted by underlying the decorated type) are defined as follows.

**Definition 1 (Ontologically refined types)** *The sets  $\mathcal{ORT}$  of ontologically refined types is the smallest sets that satisfy:*

- (1) *If  $\sigma \in \mathcal{SB}$  and  $d \in \text{Dec}_{\mathcal{O}_\sigma}$  then  $\sigma^d \in \mathcal{ORT}$  and  $\underline{\sigma^d} = \sigma$ .*
- (2) *If  $\alpha_1, \alpha_2 \in \mathcal{ORT}$  and  $d \in \text{Dec}_{\mathcal{O}_{(\alpha_1, \alpha_2)}}$ , then  $(\alpha_1, \alpha_2)^d \in \mathcal{ORT}$  and  $\underline{(\alpha_1, \alpha_2)^d} = (\underline{\alpha_1}, \underline{\alpha_2})$ .*

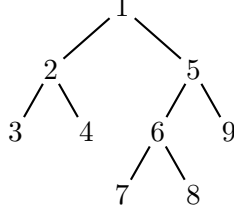
In the sequel, whenever a type is not explicitly decorated, it is assumed that the decoration is  $\top_\sigma$ .

If  $\alpha = \sigma^d$  for some  $\sigma \in \mathcal{SB}$ , then  $D_\alpha$ , the *domain of interpretation* of  $\alpha$ , is defined iff  $d \in C_{\mathcal{O}_\sigma}$ , in which case  $D_\alpha = \llbracket d \rrbracket$ . For a functional type  $\alpha = (\alpha_1, \alpha_2)^d$ ,  $D_\alpha$  is defined iff both  $D_{\alpha_1}$  and  $D_{\alpha_2}$  are defined and  $d \in C_{\mathcal{O}_{(\alpha_1, \alpha_2)}}$ , in which case  $D_\alpha \subseteq D_{\alpha_2}^{D_{\alpha_1}}$  (the inclusion is proper iff  $d < \top$ ). Furthermore, if  $\alpha' = (\alpha_1, \alpha_2)^{d'}$  for  $d' \in C_{\mathcal{O}_{(\alpha_1, \alpha_2)}}$ , then if  $d \leq d'$ ,  $D_\alpha \subseteq D_{\alpha'}$ . Otherwise  $d$  and  $d'$  are incomparable, in which case  $D_\alpha \cap D_{\alpha'} = \emptyset$ . In this paper, the matters are quite simple: we only use decorations for type  $e$ . Thus, for example,  $D_{e^{person}} = \llbracket person \rrbracket$ . More generally, if all decorations in  $\alpha_1$  and  $\alpha_2$  are equal to  $\top$  and  $d \in C_{\mathcal{O}_{(\alpha_1, \alpha_2)}}$ , then  $D_{(\alpha_1, \alpha_2)^d} = \llbracket d \rrbracket$ .

Every  $\alpha \in \mathcal{ORT}$  can be uniquely represented by a binary tree,  $T_\alpha$ , isomorphic to the construction tree of  $\underline{\alpha}$ , and two mappings: **type** $_\alpha$  from nodes in  $T_\alpha$  to subtypes of  $\alpha$ , and **dec** $_\alpha$  from nodes in  $T_\alpha$  to decorations in  $\alpha$  (subscripts will be omitted whenever possible). The nodes of the tree are numbered consecutively, starting from 1, in a depth first order. For every node  $i$  in  $T_\alpha$ , **type** $_\alpha(i)$  is the subtype of  $\alpha$  represented by the subtree of  $T_\alpha$  with root  $i$ , and **dec** $_\alpha(i)$  is the decoration **type** $(i)$ .

**Example 2** *Let  $\alpha = ((b_1^{c_1}, b_2)^{X[\leq c_2]}, ((b_1, b_2^{c_3}), b_1)^Y)$ . Then  $T_\alpha$  is the following tree,*

in which, for instance,  $\mathbf{type}(1) = \alpha$ ,  $\mathbf{type}(5) = ((b_1, b_2^{c_3}), b_1)^Y$  and  $\mathbf{type}(3) = b_1^{c_1}$ , etc.;  $\mathbf{dec}(5) = Y$ ,  $\mathbf{dec}(2) = X[\leq c_2]$ ,  $\mathbf{dec}(1) = \mathbf{dec}(4) = \top$ , etc.



**Definition 3 (Substitution list)** Let  $\alpha, \beta \in \mathcal{ORT}$  be such that  $\underline{\alpha} = \underline{\beta}$ . The substitution list  $l_{\alpha:\beta}$  is the list of pairs  $(X, c)$  such that  $X$  is an ontological variable,  $c$  a concept, and for some node  $i$  in  $T_\alpha (=T_\beta)$ ,  $\mathbf{dec}_\beta(i) = c$  and either  $\mathbf{dec}_\alpha(i) = X$  or  $\mathbf{dec}_\alpha(i) = X[\leq c']$  ( $\mathbf{dec}_\alpha(i) = X[\geq c']$ ) for some  $c' \geq c$  ( $c' \leq c$ , respectively).

A substitution list is inconsistent if it contains two pairs  $(X, c)$  and  $(X, c')$ , where  $c \neq c'$ . Otherwise the list is consistent.

An application of a function of type  $\alpha$  to function of type  $\beta$  is possible<sup>4</sup> only if  $\alpha = (\alpha_1, \alpha_2)^d$  for some  $\alpha_1, \alpha_2 \in \mathcal{ORT}$  and  $d \in \text{Dec}_{\mathcal{O}(\underline{\alpha_1}, \underline{\alpha_2})}$ , such that  $\alpha_1$  matches  $\beta$  according to the following definition.

**Definition 4 (Type matching)** Let  $\alpha, \beta \in \mathcal{ORT}$ .  $\alpha$  matches  $\beta$  iff  $\underline{\alpha} = \underline{\beta}$ ,  $l_{\alpha:\beta}$  is consistent, and for every node  $i$  in  $T_\alpha (=T_\beta)$  one of the following holds:

- (1)  $\mathbf{dec}_\alpha(i)$  is a concept and  $\mathbf{dec}_\beta(i)$  is either a concept, where  $\mathbf{dec}_\alpha(i) \geq \mathbf{dec}_\beta(i)$ , or a variable  $(X, X[\leq Y], X[\geq Y], X[\leq c]$  or  $X[\geq c])$ .
- (2)  $\mathbf{dec}_\alpha(i)$  is a variable of the form  $X[\leq c]$  ( $X[\geq c]$ ) and  $\mathbf{dec}_\beta(i)$  is a concept  $c'$  such that  $c' \leq c$  ( $c' \geq c$ , respectively).
- (3)  $\mathbf{dec}_\alpha(i)$  is a variable of the form  $X$ .<sup>5</sup>

Given a consistent substitution list  $l_{\alpha:\beta}$  and a type  $\gamma \in \mathcal{ORT}$ , we define a substitution of the concepts from  $l_{\alpha:\beta}$  for the respective variables that occur in  $\gamma$  as follows.

**Definition 5 (Substitution for ontological variables in types)** Let  $l_{\alpha:\beta} = \langle (X_1, c_1), \dots, (X_n, c_n) \rangle$  be a consistent substitution list and let  $\gamma \in \mathcal{ORT}$ .  $l_{\alpha:\beta}(\gamma)$  results from  $\gamma$  by the following operations, under the given conditions. If at some point a condition is not met, then  $l_{\alpha:\beta}(\gamma)$  is not defined (the operations are for all  $i, j$  such that  $1 \leq i, j \leq n$ , and it is assumed that

<sup>4</sup> We assume without loss of generality that  $\alpha$  and  $\beta$  do not share ontological variables.

<sup>5</sup> Note that if  $\mathbf{dec}_\alpha(i) = X[\leq c_1]$  and  $\mathbf{dec}_\beta(i) = Y[\leq c_2]$  (or  $Y[\geq c_2]$ , etc.) then there is no matching. This would require a richer language of constraints on ontological variables e.g., with decorations of the form  $X[c_2 \leq X \leq c_1]$ .

$Y \notin \{X_1, \dots, X_n\}$ .

- (1) Replace every decoration of the form  $X_i[\leq X_j]$  ( $X_i[\geq X_j]$ ) in  $\gamma$  with  $c_i$ .  
**Condition:**  $c_i \leq c_j$  ( $c_i \geq c_j$ , respectively).
- (2) Replace every decoration of the forms  $X_i[\leq c]$  ( $X_i[\geq c]$ ) in  $\gamma$  with  $c_i$ .  
**Condition:**  $c_i \leq c$  ( $c_i \geq c$ , respectively)
- (3) Replace every decoration of one of the forms  $X_i$ ,  $X_i[\leq Y]$  and  $X_i[\geq Y]$  with  $c_i$ .
- (4) Replace every decoration of the form  $Y[\geq X_i]$  ( $Y[\leq X_i]$ ) with  $Y[\geq c_i]$  ( $Y[\leq c_i]$ , respectively).

We are now in a position to define exactly when a function of type  $\alpha$  can apply to a function of type  $\beta$ .

**Definition 6 (Type application)** Let  $\alpha, \beta \in \mathcal{ORT}$ .  $\mathbf{app}[\alpha, \beta] = \gamma$  iff  $\alpha = (\alpha_1, \alpha_2)$  for some  $\alpha_1, \alpha_2 \in \mathcal{ORT}$  such that  $\alpha_1$  matches  $\beta$  and  $\gamma = l_{\alpha_1; \beta}(\alpha_2)$ .

**Example 7** Consider the ontology in figure 1. Then:

- $\mathbf{app}[(e^{\text{animate}}, t), ((e, t), t)], (e^{\text{person}}, t)] = ((e, t), t)$ .
- $\mathbf{app}[(e^{\text{animate}}, t), ((e, t), t)], (e^{\text{inanimate}}, t)]$  is undefined, since  $\text{inanimate} \not\leq \text{animate}$ .
- $\mathbf{app}[(e^X, t), ((e^{Y[\geq X]}, t), t)], (e^{\text{person}}, t)] = ((e^{Y[\geq \text{person}]}, t), t)$ .
- $\mathbf{app}[(e^X, t), e^X] = \mathbf{app}[(e^{Y[\geq X]}, t), e^X] = \mathbf{app}[(e^{Y[\leq X]}, t), e^X] = t$ .

For every  $\alpha \in \mathcal{ORT}$ , let  $Var_\alpha$  and  $Con_\alpha$  be countable sets of term-variables and constants (respectively) of type  $\alpha$ . Every  $x \in Var_\alpha$  will be written as  $x^\alpha$ , and every  $a \in Con_\alpha$  – as  $a^\alpha$ . The set  $OTerm_\alpha$  of (ontologically well-typed)  $\lambda$ -terms of type  $\alpha \in \mathcal{ORT}$ , is defined as follows.

- Definition 8 (Ontologically well-typed  $\lambda$ -terms)**
- (1)  $Var_\alpha \cup Con_\alpha \subseteq OTerm_\alpha$ .
  - (2) Let  $F \in OTerm_{(\alpha_1, \alpha_2)^d}$  and  $A \in OTerm_\beta$  s.t.  $\mathbf{app}[(\alpha_1, \alpha_2)^d, \beta] = \alpha$ . Let  $F' = l_{\alpha_1; \beta}(F)$ <sup>6</sup> and  $A' = l_{\alpha_1; \beta}(A)$ . Then  $(F' A') \in OTerm_\alpha$  iff both  $F'$  and  $A'$  are ontologically well-typed.
  - (3) If  $P \in OTerm_{\alpha_2}$ ,  $x \in Var_{\alpha_1}$  and  $d \in Dec_{\mathcal{O}(\alpha_1, \alpha_2)}$ , then  $(\lambda x.P)^d \in OTerm_{(\alpha_1, \alpha_2)^d}$  (when  $d = \top$  it is usually omitted).

<sup>6</sup> Given a substitution list  $l_{\sigma; \sigma'}$  and a  $\lambda$ -term  $M$ ,  $l(M)$  result by a simultaneous substitution of ontological variables in the types of all term-variables and constants in  $M$ .

Note that  $l(M)$  need not be ontologically well-typed, even when  $M$  itself is ontologically well-typed. Example: if  $l = \langle (X : c_1) \rangle$  and  $c_1 < c_2$ , then  $(x^{(\sigma_1^X, \sigma_2)} y^{\sigma_1^{c_2}})$  is well-typed, while  $l((x^{(\sigma_1^X, \sigma_2)} y^{\sigma_1^{c_2}})) = (x^{(\sigma_1^{c_1}, \sigma_2)} y^{\sigma_1^{c_2}})$  is not.

**Definition 9 ( $\beta$ -reduction for ontologically-well typed  $\lambda$ -terms)** Let  $P \in OTerm_\gamma$ , s.t.  $P = ((\lambda x^{\alpha_1}. M)^d N)$  for some  $M \in OTerm_{\alpha_2}$ ,  $N \in OTerm_\beta$  and  $d \in Dec_{\mathcal{O}(\alpha_1, \alpha_2)}$  (thus,  $\mathbf{app}[(\alpha_1, \alpha_2)^d, \beta] = \gamma$ ). Let  $Q = [l_{\alpha_1: \beta}(N)/x]l_{\alpha_1: \beta}(M)$ , where  $N$  is free for  $x$  in  $M$ . Then  $P \rightarrow_\beta Q$ .

The assumption that  $P$  is ontologically well-typed does not entail that  $Q$  is also ontologically well-typed. For example, assume  $c_1 > c_2 > c_3$ , and  $P = ((\lambda x^{(\alpha_1^{c_1}, \alpha_2)}. xy^{\alpha_1^{c_2}})z^{(\alpha_1^{c_3}, \alpha_2)})$ . Then  $P \rightarrow_\beta Q$  for  $Q = (z^{(\alpha_1^{c_3}, \alpha_2)} y^{\alpha_1^{c_2}})$ , which is certainly not ontologically well-typed.

**Example 10** An application of the  $\lambda$ -term assigned to every in table 1 to the  $\lambda$ -term assigned to boy:

$$\begin{aligned} & (\lambda A^{(e^X, t)} \lambda B^{(e^Y[\geq X], t)} \forall x^{e^X}. (Ax) \rightarrow (Bx)) \lambda y^{e^{\text{person}}}. (\mathbf{boy}^{(e^{\text{person}}, t)} y)) \rightarrow_\beta \\ & \lambda B^{(e^Y[\geq \text{person}], t)} \forall x^{e^{\text{person}}}. (\lambda y^{e^{\text{person}}}. (\mathbf{boy}^{(e^{\text{person}}, t)} y)) x) \rightarrow (Bx) \rightarrow_\beta \\ & \lambda B^{(e^Y[\geq \text{person}], t)} \forall x^{e^{\text{person}}}. (\mathbf{boy}^{(e^{\text{person}}, t)} x) \rightarrow (Bx) \end{aligned}$$

Note how all occurrences of  $X$  in the scope of  $\lambda A^{(e^X, t)}$  are instantiated to ‘person’ in the first  $\beta$ -reduction step.

Syntactic categories are inductively defined, based on a finite non-empty set  $\mathcal{BC}$  of basic categories.

**Definition 11 (Categories)** The set  $\mathcal{C}$  of categories is the smallest set that satisfies:

- (1)  $\mathcal{BC} \subseteq \mathcal{C}$ .
- (2) If  $\tau_1, \tau_2 \in \mathcal{C}$  then  $(\tau_1 \leftarrow \tau_2) \in \mathcal{C}$  and  $(\tau_1 \rightarrow \tau_2) \in \mathcal{C}$ .

## 4 The calculus $\mathbf{L}^\mathcal{O}$

In this section we introduce a modification of  $\mathbf{L}$ , the Ontological Lambek calculus ( $\mathbf{L}^\mathcal{O}$ ), that make use of the types in  $\mathcal{ORT}$ .  $\mathbf{L}^\mathcal{O}$  differs from  $\mathbf{L}$  in that the arrow elimination rules respect the definition of type application (definition 6). Note that if only standard types are considered then  $\mathbf{L}^\mathcal{O}$  is equivalent to  $\mathbf{L}$  (follows from a similar property of  $\mathbf{app}$ ).

The ultimate goal is that if  $G$  is a grammar that is based on  $\mathbf{L}^\mathcal{O}$ , then every sentence in the language of  $G$ ,  $L(G)$ , is both grammatical and semantically

well-formed, with respect to the given ontologies. For example, in the small grammar that will be defined in section 5 below, the sentence in (1a) is derivable, while the grammatical but semantically bad sentence in (1b) is not.

The basic units of the calculus are pairs  $\tau : M^\alpha$ , where  $\tau \in \mathcal{C}$ ,  $\alpha \in \mathcal{ORT}$  and  $M \in \mathcal{OTerm}_\alpha$ . We use meta variables  $\Gamma$  (possibly subscripted) to range over nonempty sequences of such pairs, in which each  $M$  is a term variable. Given a substitution list  $l$  and a sequence  $\Gamma = \tau_1 : x_1^{\alpha_1} \cdots \tau_n : x_n^{\alpha_n}$ , we write  $l(\Gamma) = \tau_1 : x_1^{l(\alpha_1)} \cdots \tau_n : x_n^{l(\alpha_n)}$ . A *sequent* is a formula of the form  $\Gamma \triangleright \tau : M^\alpha$ , expressing, as usual, *reducibility* of  $\Gamma$  to  $\tau : M^\alpha$ . Given a calculus  $L$ , the statement  $\vdash_L \Gamma \triangleright \tau : M^\alpha$  indicates that the sequent  $\Gamma \triangleright \tau : M^\alpha$  is provable in  $L$ .

**Definition 12 (The  $L^O$  calculus)** *Axioms:*  $\tau : A^\alpha \triangleright \tau : A^\alpha$ , for every  $\tau \in \mathcal{C}$ ,  $\alpha \in \mathcal{ORT}$  and  $A \in \mathcal{Var}_\alpha$ .

The arrow elimination rules are as follows, where  $\mathbf{app}[(\alpha_1, \alpha_2), \alpha] = \beta$ ,  $\Gamma'_1 = l_{\alpha_1:\alpha}(\Gamma_1)$ ,  $\Gamma'_2 = l_{\alpha_1:\alpha}(\Gamma_2)$ ,  $F' = l_{\alpha_1:\alpha}(F)$  and  $A' = l_{\alpha_1:\alpha}(A)$ .

$$\frac{\Gamma_1 \triangleright \tau_1 : A^\alpha \quad \Gamma_2 \triangleright (\tau_1 \rightarrow \tau_2) : F^{(\alpha_1, \alpha_2)}}{\Gamma'_1 \Gamma'_2 \triangleright \tau_2 : (F' A')^\beta} (\rightarrow E) \qquad \frac{\Gamma_1 \triangleright (\tau_2 \leftarrow \tau_1) : F^{(\alpha_1, \alpha_2)} \quad \Gamma_2 \triangleright \tau_1 : A^\alpha}{\Gamma'_1 \Gamma'_2 \triangleright \tau_2 : (F' A')^\beta} (\leftarrow E)$$

The arrow introduction rules are as follows, where  $\Gamma$  is not empty and  $d \in \mathcal{Dec}_{\mathcal{O}(\underline{\alpha_1, \alpha_2})}$ .

$$\frac{\tau_1 : x^{\alpha_1}, \Gamma \triangleright \tau_2 : M^{\alpha_2}}{\Gamma \triangleright (\tau_1 \rightarrow \tau_2) : (\lambda x.M)^{(\alpha_1, \alpha_2)}} (\rightarrow I) \qquad \frac{\Gamma, \tau_1 : x^{\alpha_1} \triangleright \tau_2 : M^{\alpha_2}}{\Gamma \triangleright (\tau_1 \leftarrow \tau_2) : (\lambda x.M)^{(\alpha_1, \alpha_2)}} (\leftarrow I)$$

Regarding the arrow elimination rules, recall that **app** may have side-effects, instantiating concept variables in the premises of the arrow-application rule. The notation here means that the side-effects are reflected also in the resulting sequent. For example:

$$\frac{(\tau_1 \leftarrow \tau_2) : x_1^{(\alpha_1^X, \alpha_2^{Y[\geq X]})} \triangleright (\tau_1 \leftarrow \tau_2) : x_1^{(\alpha_1^X, \alpha_2^{Y[\geq X]})} \quad \tau_2 : x_2^{\alpha_1^c} \triangleright \tau_2 : x_2^{\alpha_1^c}}{(\tau_1 \leftarrow \tau_2) : x_1^{(\alpha_1^c, \alpha_2^{Y[\geq c]})}, \tau_2 : x_2^{\alpha_1^c} \triangleright \tau_1 : x_1^{(\alpha_1^c, \alpha_2^{Y[\geq c]})} (x_2^{\alpha_1^c})} (\leftarrow E)$$

In practice, we shall restrict ourselves to proofs that are in normal form. An example of a non-normalized proof is the following, in which  $c, c' \in C_{\mathcal{O}_\alpha}$  s.t.  $c < c'$ .

$$\frac{(\tau_2 \leftarrow \tau_1) : x_1^{(\alpha^{c'}, \beta)} \triangleright (\tau_2 \leftarrow \tau_1) : x_1^{(\alpha^{c'}, \beta)} \quad \tau_1 : x_2^{\alpha^{c'}} \triangleright \tau_1 : x_2^{\alpha^{c'}}}{(\tau_2 \leftarrow \tau_1) : x_1^{(\alpha^{c'}, \beta)}, \tau_1 : x_2^{\alpha^{c'}} \triangleright \tau_2 : (x_1(x_2))^{\beta}} (\leftarrow E)$$

$$\frac{(\tau_2 \leftarrow \tau_1) : x_3^{(\alpha^c, \beta)} \triangleright (\tau_2 \leftarrow \tau_1) : x_3^{(\alpha^c, \beta)} \quad \tau_1 : x_2^{\alpha^{c'}} \triangleright ((\tau_2 \leftarrow \tau_1) \rightarrow \tau_2) : (\lambda x_1.x_1(x_2))^{((\alpha^{c'}, \beta), \beta)}}{(\tau_2 \leftarrow \tau_1) : x_3^{(\alpha^c, \beta)}, \tau_1 : x_2^{\alpha^{c'}} \triangleright \tau_2 : (x_3(x_2))^{\beta}} (\rightarrow E)$$

Of course, we want a term of type  $(\alpha^c, \beta)$  to be applicable to a term of type  $\alpha^{c'}$  iff  $c \geq c'$ .

## 5 Grammar

In this section we consider an example grammar. We assume  $\mathcal{SB} = \{e, t\}$  and  $\mathcal{BC} = \{n, np, s\}$ . In addition, let  $\mathcal{O}_e$  be the ontology in Figure 1.

The small lexicon in Table 1 consists of determiners, nouns, intransitive and transitive verbs, and adjectives. For example, the sentence in (1a) can be derived, while the bad sentence in (1b) cannot. Similarly, (2a) is a verb phrase in the grammar (category  $(np \rightarrow s)$ ) but (2b) is not. Also, (3a) is of category  $n$ , with *exactly* the same type as **table**; (3b) cannot be proved to be of category  $n$ . Following are examples of derivations in  $\mathbf{L}^{\mathcal{O}}$ . In these examples, discharged assumptions are in square brackets and numbered for readability, as in  $\mathbf{L}$ . Also, types are shown only for full  $\lambda$ -term to save space. For more detailed presentation of the  $\lambda$ -terms, refer to table 1.

To fit the derivations to the size of the page, we let  $\Delta[\mathbf{every}(\mathbf{table})]$  and  $\Delta[\mathbf{every}(\mathbf{boy})]$  be the following (sub)derivations.

$$\begin{array}{c}
 \frac{\frac{\text{every}}{\frac{((s \leftarrow np) \rightarrow s) \leftarrow n) :}{(\lambda A \lambda B. \forall x. A(x) \rightarrow B(x))^{((e^X, t), ((e^Y[\geq X], t), t))}} \quad (lex) \quad \frac{\text{table}}{\frac{n :}{(\lambda x. \mathbf{table}(x))^{(e^{furniture}, t)}}} \quad (lex)}{\frac{((s \leftarrow np) \rightarrow s) :}{(\lambda B. \forall x. \mathbf{table}(x) \rightarrow B(x))^{(e^Y[\geq furniture], t), t)}}} \quad (\leftarrow E)
 \end{array}$$
  

$$\begin{array}{c}
 \frac{\frac{\text{every}}{\frac{((s \leftarrow np) \rightarrow s) \leftarrow n) :}{(\lambda A \lambda B. \forall x. A(x) \rightarrow B(x))^{((e^X, t), ((e^Y[\geq X], t), t))}} \quad (lex) \quad \frac{\text{boy}}{\frac{n :}{(\lambda x. \mathbf{boy}(x))^{(e^{person}, t)}}} \quad (lex)}{\frac{((s \leftarrow np) \rightarrow s) :}{(\lambda B. \forall x. \mathbf{boy}(x) \rightarrow B(x))^{(e^Y[\geq person], t), t)}}} \quad (\leftarrow E)
 \end{array}$$

The first example shows that (for expositional purposes, syntactic categories are omitted here):

$$\vdash_{\mathbf{L}^{\mathcal{O}}} ((e^X, t), ((e^Y[\geq X], t), t)), (e^{person}, t), (e^{animate}, t) \triangleright t$$

Hence, sentence (1a) has type  $t$  in our grammar, with its usual meaning.

$$\begin{array}{c}
\frac{\text{smiled}}{(np \rightarrow s) :} \text{ (lex)} \\
\frac{\Delta[\text{every}(\text{boy})] \quad (\lambda x.\text{smile}(x))^{(e^{animate}, t)}}{s :} \text{ (}\leftarrow E\text{)} \\
(\forall x.\text{boy}(x) \rightarrow \text{smile}(x))^t
\end{array}$$

On the other hand, the string in (1b), in spite of being syntactically well-formed (of category  $s$ ), does not have type  $t$  in our grammar. This fact is demonstrated by the following unsuccessful attempt to prove that:

$$\vdash_{\mathbf{L}^\circ} ((e^X, t), ((e^{Y[\geq X]}, t), t)), (e^{furniture}, t), (e^{animate}, t) \triangleright t$$

The derivation fails because  $furniture \not\leq animate$ .

$$\begin{array}{c}
\frac{\text{smiled}}{(np \rightarrow s) :} \text{ (lex)} \\
\frac{\Delta[\text{every}(\text{table})] \quad (\lambda x.\text{smile}(x))^{(e^{animate}, t)}}{FAIL} \text{ (}\leftarrow E\text{)}
\end{array}$$

It should be noted that the fact that this derivation is unsuccessful is certainly not a proof that:

$$\not\vdash_{\mathbf{L}^\circ} ((e^X, t), ((e^{Y[\geq X]}, t), t)), (e^{furniture}, t), (e^{animate}, t) \triangleright t$$

This statement is provable, but the proof is tedious. Of course, the same remark applies to the other unsuccessful derivations below as well.

The next two examples pertain to the connection between a transitive verb and its object.

The first of the two examples shows that:

$$\vdash_{\mathbf{L}^\circ} (e^\top, (e^{animate}, t)), ((e^X, t), ((e^{Y[\geq X]}, t), t)), (e^{furniture}, t) \triangleright (e^{animate}, t)$$

In this proof  $c \in C_{\mathcal{O}_e}$  is any concept such that  $c \geq furniture$ . Hence, the VP saw every table has type  $(e^{animate}, t)$ . Thus, every idea saw every table is not derivable.

$$\begin{array}{c}
\frac{\text{say}}{\text{((np} \rightarrow \text{s)} \leftarrow \text{np}) :} \quad (\text{lex}) \\
\frac{\text{((np} \rightarrow \text{s)} \leftarrow \text{np}) :} {(\lambda x \lambda y. \text{see}(x)(y))^{(e^\top, (e^{\text{animate}}, t))}} \quad \left[ \begin{array}{l} \text{np} : \\ z^{e^c} \end{array} \right]_1 \\
\hline
\frac{}{\text{(np} \rightarrow \text{s}) :} \quad (\leftarrow E) \\
\frac{\text{(np} \rightarrow \text{s}) :} {(\lambda y. \text{see}(z)(y))^{(e^{\text{animate}}, t)}} \quad (\rightarrow E) \\
\hline
\text{s :} \\
\frac{\text{(see}(z)(v))^t}{} \quad (\leftarrow I_1) \\
\frac{\text{(s} \leftarrow \text{np}) :} {(\lambda z. \text{see}(z)(v))^{(e^c, t)}} \quad \Delta[\text{every}(\text{table})] \\
\hline
\text{s :} \\
\frac{\text{(}\forall y. \text{table}(y) \rightarrow \text{see}(y)(v))^t}{} \quad (\rightarrow I_2) \\
\frac{\text{(}\forall y. \text{table}(y) \rightarrow \text{see}(y)(v))^t} {(\lambda v. (\forall y. \text{table}(y) \rightarrow \text{see}(y)(v)))^{(e^{\text{animate}}, t)}} \quad (\rightarrow E)
\end{array}$$

On the other hand, the next example demonstrates why:

$$(e^{\text{animate}}, (e^{\text{animate}}, t)), ((e^X, t), ((e^{Y[\geq X]}, t), t)), (e^{\text{furniture}}, t) \triangleright (e^{\text{animate}}, t)$$

Note that  $c_1$  must satisfy  $c \leq \text{animate}$ . Otherwise the  $(\leftarrow E)$  step with the axiom  $\text{np} : z^{e^c}$  as one of the premisses would fail. This would entail a failure in the last  $(\rightarrow E)$  step, because then  $c, c \not\leq \text{furniture}$ . Accordingly, the string *met every table* is not derivable as a VP in our grammar.

$$\begin{array}{c}
\frac{\text{met}}{\text{((np} \rightarrow \text{s)} \leftarrow \text{np}) :} \quad (\text{lex}) \\
\frac{\text{((np} \rightarrow \text{s)} \leftarrow \text{np}) :} {(\lambda x \lambda y. \text{meet}(x)(y))^{(e^{\text{animate}}, (e^{\text{animate}}, t))}} \quad \left[ \begin{array}{l} \text{np} : \\ z^{e^c} \end{array} \right]_1 \\
\hline
\frac{}{\text{(np} \rightarrow \text{s}) :} \quad (\leftarrow E) \\
\frac{\text{(np} \rightarrow \text{s}) :} {(\lambda y. \text{meet}(z)(y))^{(e^{\text{animate}}, t)}} \quad (\rightarrow E) \\
\hline
\text{s :} \\
\frac{\text{(meet}(z)(v))^t}{} \quad (\leftarrow I_1) \\
\frac{\text{(s} \leftarrow \text{np}) :} {(\lambda z. \text{meet}(z)(v))^{(e^c, t)}} \quad \Delta[\text{every}(\text{table})] \\
\hline
\text{FAIL} \quad (\rightarrow E)
\end{array}$$

We end this section with two more examples, regarding adjectival modification. The first example shows that:

$$\vdash_{\mathbf{L}^\circ} ((e^{X[\leq \text{material}]}, t), (e^X, t)), (e^{\text{furniture}}, t) \triangleright (e^{\text{furniture}}, t)$$

while the second demonstrates why:

$$\not\vdash_{\mathbf{L}^\circ} ((e^{X[\leq \text{material}]}, t), (e^X, t)), (e^{\text{cognitive}}, t) \triangleright (e^{\text{cognitive}}, t)$$

because *cognitive*  $\not\leq$  *material*.

$$\begin{array}{c}
\frac{\text{red} \quad (lex)}{(n \leftarrow n) :} \quad \frac{\text{table} \quad (lex)}{n :} \\
\frac{(\lambda x.\text{red}(x))^{(e^{X[\leq \text{material}],t},(e^X,t))} \quad (\lambda x.\text{table}(x))^{(e^{\text{furniture}},t)}}{n :} \quad (\leftarrow E) \\
\text{red}(\text{table})^{(e^{\text{furniture}},t)}
\end{array}$$

$$\begin{array}{c}
\frac{\text{red} \quad (lex)}{(n \leftarrow n) :} \quad \frac{\text{idea} \quad (lex)}{n :} \\
\frac{(\lambda x.\text{red}(x))^{(e^{X[\leq \text{material}],t},(e^X,t))} \quad (\lambda x.\text{idea}(x))^{(e^{\text{cognitive}},t)}}{FAIL} \quad (\leftarrow E)
\end{array}$$

## 6 Conclusions

In this paper we have presented a refinement of the standard semantic types for TLG. The type system is enriched by way of decorating types with ontological concepts or variables. Accordingly, the conditions that determines when a term of type  $\alpha$  can apply to a term of type  $\beta$  are more complex than in standard TLG. The decorated types and the conditions for type application plays a central role both in the definition of ontologically well typed  $\lambda$ -terms, and in the definition of the calculus  $\mathbf{L}^{\mathcal{O}}$ .

The main purposes of the formalism presented here is to enable the definition of grammars in which semantically bad sentences would not be derived, even if the sentences are syntactically well-formed, while maintaining the mathematical elegance of standard TLGs as much as possible.

We presented a small grammar based on the formalism. In this grammar we used ontology only for type  $e$ . We have shown how selectional restrictions can be imposed in the grammar, using decorated types, so that semantically bad sentences would not be derived. A natural question that arises in this context is how to deal with linguistic phenomenon other than those that presented here, namely restrictive adjectival modification, and selectional restrictions that a verb imposes on its NP arguments. We expect that with a richer lexicon, ontologies for types other than  $e$  will be required. We leave this issue for further research.

From a logical point of view, a general question is the following. Given a property of standard Lambek calculus, is that property maintained by the extended formalism? Specifically, what is the connection of the extended formalism to the Curry-Howard correspondence? Also, recall that we note at the end of section 4 that we consider only proofs in normal form. This restriction and its implications are subject for further research.

As for the ontologically well-types  $\lambda$ -terms, note that  $\eta$ -reduction does not necessarily preserve types. For example, if  $P$  is of type  $(e^{c_1}, t)$ , then for any concept  $c_2$  such that  $c_2 \leq c_1$ , and for any  $x$  of type  $e^{c_2}$ ,  $\lambda x.(Px)$  is of type  $(e^{c_2}, t)$ . This has been noticed in [9], and related to the *restrictions* of functions to subdomains.

## References

- [1] M. Moortgat, Categorical type logics, in: J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, Elsevier, Amsterdam and MIT Press, Cambridge MA, 1997, pp. 93–177.
- [2] R. Montague, English as a formal language, in: B. Visentini, et al. (Eds.), Linguaggi nella Società e nella Tecnica, Edizioni di Comunità, Milan, 1970, reprinted in [10].
- [3] R. Montague, The proper treatment of quantification in ordinary english, in: J. Hintikka, J. Moravcsik, P. Suppes (Eds.), Approaches to Natural Language, Reidel, Dordrecht, 1973, reprinted in [10].
- [4] J. Pustejovsky, Type construction and the logic of concepts, in: P. Bouillon, F. Busa (Eds.), The Language of Word Meaning, Cambridge University Press, 2001, pp. 91–123.
- [5] P. Cimiano, U. Reyle, Ontology-based semantic construction underspecification and disambiguation, in: Proceedings of the Lorraine/Saarland Workshop on Prospects and Recent Advances in the Syntax-Semantics Interface, October 20–21, 2003, Nancy, France, 2003.  
URL <http://www.aifb.uni-karlsruhe.de/WBS/pci/prospects.pdf>
- [6] P. A. Jensen, J. F. Nilsson, Ontology-based semantics for prepositions, in: ACM-SIGSEM Workshop on the Linguistic Dimension of Prepositions and their use in Computational Linguistics, Institut de Recherche en Informatique de Toulouse, Toulouse, September 4–6, 2003. UMR 5505 CNRS-INP-UPS, 2003.
- [7] J. Lambek, The mathematics of sentence structure, American Mathematical Monthly 65 (1958) 154–170.
- [8] M. Kohlhase, F. Pfenning, Unification in a  $\lambda$ -calculus with intersection types, in: D. Miller (Ed.), Proceedings of the International Logic Programming Symposium, MIT Press, Cambridge, Massachusetts, 1993, pp. 488–505.
- [9] P. Johann, M. Kohlhase, Unification in an extensional lambda calculus with ordered function sorts and constant overloading, in: Proceedings of the 12th International Conference on Automated Deduction, CADE’94, Lecture Notes in Computer Science, 814, 1994.
- [10] R. H. Thomason (Ed.), Formal Philosophy: Selected Papers of Richard Montague, Yale University Press, New Haven, 1974.