SENSE TYPE DECISION TREES FOR NATURAL LANGUAGE PROCESING

Maria Zamfir Bleyberg
Computing and Information Sciences Department
Kansas State University
Manhattan, KS, 66506-2302
maria@cis.ksu.edu

Abstract

In this paper, we treat English texts in the same way as strongly-typed functional programming languages. We show that the English text categorization can be improved by considering *compound* semantic categories as *sense types* associated to its structural configurations. We use a *sense type decision tree* of *atomic* semantic categories together with typing axioms and inference rules, which represent meaningful relationships among semantic categories, to construct compound semantic categories, validate semantic category hypotheses, and solve ambiguities.

Keywords: text mining, compound categorization, sense type decision trees, ambiguity, typed lambda-calculus.

1 Introduction

In recent years, we have witnessed an immense growth in the availability of on-line information. Given that much of this information is textual in nature, a complex methodology is necessary to retrieve *only* those documents that are requested by the user. Simple text categorization [3, 2] is one paradigm that is frequently used to classify documents. A basic problem with simple text categorization is that the assignment of one or more labels from a pre-defined set to a document assumes the independence of the words.

In processing of functional programming languages, the concept and role of types is well established. For such artificial languages, there are commonly accepted atomic types and constructors for new types [4]. Sentences of the language are formed according to well-defined rules of grammar and type consistency.

Intensive work has also been done on the logical development of the categorial approach to natural language analysis. In [5, 10], syntactic categories reflect distributional phenomena in natural languages and types play the role of their semantic counterparts. In our work, the emphasis is on semantic categories for document exploration.

In this paper, we construct a binary *sense type decision tree* having *atomic* sense types (i.e., semantic categories) as nodes and use it to derive *compound* semantic categories under given composition rules. The composition rules capture the *meaningful* relationships among semantic categories. Following the treatment of strongly-typed functional programming languages used in [4], we define a logic by introducing axioms to represent *atomic* semantic categories and inference rules to derive *compound* semantic categories. The tree proofs that can be constructed in this proof system validate compound semantic category hypotheses and help solve subtle *ambiguities* that occur when a sentence fragment has more than one semantic category associated to it. Compound categories increase the precision of document exploration.

Decision trees are among the most popular classification techniques currently used. Classification depends of a given a set of cases, called a *training set*, where each case consists of several fields or attributes. One of the attributes, called the *classifying* attribute, indicates the *class* to which each case belongs. The objective of classification is to build a predictive model of the classifying attribute based upon the other attributes. The root of the decision tree is associated with the set of all training cases. All other nodes are associated with some subset of the cases determined by the path from the root to the node. The partition of the cases at a node into subsets is done according to the values of a chosen attribute. A node is terminal if all the cases associated with that node fall in a single class.

A decision tree classifies a new case by starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the case's outcome for the test at the node is determined and the move is directed to the root of the subtree corresponding to this outcome. When a leaf is reached, the class of the case is predicted to be that recorded at the leaf.

An attempt to predict whether a new case will show a certain type of behavior in fact implies an assumption that the case belongs to a certain class and will therefore show this certain kind of behavior.

In our work, the root of the sense type decision tree is associated with a universe of English nouns from a given collection of documents. The role of attributes is played by adjectives and verbs. Following the idea presented in Sommers' sense logic [9], we distinguish *two* kinds of sentences: **meaningful** ones and **meaningless** ones. An adjective or verb associated with a noun generates either a **meaningful** sentence or a **meaningless** sentence. Therefore, we need to consider only two classes for our cases.

The construction of a sense type decision tree leads to the word-sense disambiguation: a distinctive sense type is associated to each meaning of a noun. A corpus-based approach to the word-sense disambiguation problem can be found in the work of Pedersen and Bruce [6].

The present work is a continuation of the work reported in [1], in which a *sense typed* subset of English has been introduced using strongly-typed lambda calculus.

2 Sense Type Decision Trees

The most common way of defining all the legal structures of a language is by using a *context-free* grammar. We present in Figure 1 the syntax rules of a simple context-free grammar for the English sentences in Figure 2 using the BNF format.

According to Sommers' sense logic [9], natural language sentences fall into *two* classes: **meaningful** (**m**) and **meaningless** (**nm**). Based on this idea, we construct a *binary* sense type decision tree by associating *sense types* to sentence structures. A universe of English nouns and pronouns is associated with the root of the decision tree. Selected attributes, which are adjectives or verbs of the lexicon, are used to split the set of nouns at a node into *complimentary* sets.

We choose the attribute A with the highest *entropy* for splitting the set of nouns S at a given node [7]:

$$Entropy(S, A) \equiv -p_{\mathbf{m}} log_2 p_{\mathbf{m}} - p_{\mathbf{nm}} log_2 p_{\mathbf{nm}}$$

where $p_{\mathbf{m}}$ is the proportion of meaningful examples in S with respect to the attribute A and $p_{\mathbf{nm}}$ is the proportion of meaningless examples in S with respect to the attribute A. Notice that the *entropy* is 0 if all members of S belong to the same class. The *entropy* is 1 when the number of meaningful examples is equal to the number of meaningless examples with respect to the attribute A. The splitting process is terminated when all the attributes of the given lexicon have been considered.

Besides of a set of nouns, a set of attributes is associated with each node of the binary sense type decision tree. Such a set contains all the adjectives and verbs for which the set of nouns has entropy 0.

A binary sense type decision tree *does not* represent the classification of words in various categories, such as *green*, *red*, and *yellow* are *colors*. It shows the nouns and pronouns partitioned into complementary sets in such a way that certain adjectives and verbs can be meaningfully applied to all the nouns of a group and cannot be meaningfully applied to the nouns of the other group. Some nouns, such as *program*, may belong to several categories. An example of a binary sense type decision tree (with some instances) is given in Figure 3.

In this example, we have the node **Product** associated with the set $\{$*program*^{1}, *warehouse, workstation, desktop*$\}$. The verb *run* in conjunction with *program* and *warehouse* leads to **meaningful** expressions: *a* [*C*] *program runs* [*on a computer*], *a* [*data*] *warehouse runs* [*on a mainframe*]. The same verb leads to **meaningless** expressions in conjunction with *workstation* and *desktop*, such as, *a workstation runs.* Some nouns have several meanings. In such a case, the noun will be indexed, as in a dictionary. For example, we write *program*^{1} if *program* denotes a computer program, and *program*^{2} if *program* denotes a plan to be followed.

Formally, a binary sense type decision tree represents a nested family of *classes* (*labeled binary partitions*) on the *Universe* U of nouns (proper and common) and pronouns. The set inclusion relation $\subseteq$ on the collection $\mathcal{P}(U)$ of these classes is a partial order, which gives a tree structure to $\mathcal{P}(U)$. We call a label of a class, $L_{i_1 \cdots i_k 1}$, an *atomic* sense type. The partial order $\subseteq$ on $\mathcal{P}(U)$ induces a partial order $\sqsubseteq$ on the collection $\mathcal{L}$ of the atomic sense types of the $\mathcal{P}(U)$ classes. The following sequence is an example of a binary hierarchy:

$$L_1 \sqsubseteq L_{11} \sqsubseteq \cdots \sqsubseteq L_{i_1 \cdots i_k} \sqsubseteq L_{i_1 \cdots i_k 1}, \cdots \qquad (1)$$

Let L_i, L_j be arbitrary sense atomic types of $\mathcal{L}$. We shall write $L_i \sqsubseteq^* L_j$ to indicate that there is a finite number of chain links from L_i to L_j in the sense type decision tree. We say that L_i and L_j are *comparable* and that L_j *subsumes* L_i. We call *generalization* moving up in the tree and *specialization* moving down in the tree. For example, **Person** $\sqsubseteq^*$ **Object**, i.e., **Object** subsumes **Person**. We write $L_i \parallel L_j$ if $L_i \not\sqsubseteq L_j$ and $L_i \not\sqsupseteq L_j$, i.e., if L_i and L_j are *not comparable*.

An interactive tool for the construction of such a binary sense type decision tree has been presented in [8].

S	::=	**NP VP**
NP	::=	**PrName** | **Noun** | **NP Conj NP** | **Det Noun** | **Det Adj Noun**
VP	::=	**IVerb** | **TrVerb NP** | **VP PP**
PP	::=	**Prep NP** | **NP PP**
Noun	::=	*grads* | *warehouse* | *program(s)* | *desktop(s)* | *data* | *idea(s)* | *roof*
PrName	::=	*Oracle* | *Compaq*
TrVerb	::=	*hire(s)* | *develop(ed)* | *join(t)* | *build(ing)*
IVerb	::=	*run*
Det	::=	*a* | *the*
Adj	::=	*joint* | *training* | *green*
Prep	::=	*for* | *to* | *with*
Conj	::=	*and*

Figure 1: A simple *context-free* grammar

[*a*] *Oracle hires grads to build a warehouse with a green roof.*
[*b*] *Oracle hires grads to build a data warehouse.*
[*c*] *Oracle and Compaq developed a joint training program.*
[*d*] *Oracle and Compaq developed programs for desktops.*

Figure 2: English sentences displaying ambiguities

The sense type expressions that we consider in this work are given by the following definition. Given the set $\mathcal{L}$ of atomic types, we define sense types as follows:

- each atomic type is a sense type
- if α and β are sense types, then $(\alpha \rightarrow \beta)$ is a sense type
- if $\alpha \rightarrow \beta$ is a sense type, then $(\alpha \Rightarrow \beta)$ is a *compound* sense type.

Each atomic sense type $L \in \mathcal{L}$ denotes the class of nouns $[U]$ it represents. The connective $\Rightarrow$ will make possible to explicitly refer to compound types. Each type $(\alpha \rightarrow \beta) \rightarrow (\alpha \Rightarrow \beta)$ is intended to denote some given set of functions from the type α and the type β to $(\alpha \Rightarrow \beta)$. The interpretation of a compound type is a collection of *meaningful* sentences. A sentence is *meaningless* if no type can be assigned to it.

Underlying the interpretation of a compound type expression is the concept of a *typed lambda abstraction*, which will be presented in the next section.

3 The Lexicon

Lambda abstraction [4] has proven useful in writing *function expressions* and *application*, which allows one to make use of the functions defined. A simple example of a typed lambda expression is

$$I : \mathbf{nat} \rightarrow \mathbf{nat} \;\equiv\; (\lambda x^{\mathbf{nat}} \,.\, x^{\mathbf{nat}})^{\mathbf{nat}\rightarrow\mathbf{nat}}$$

representing the identity function on natural numbers.

We write $(\lambda x\ y\ .\ x)^{\alpha\rightarrow\beta\rightarrow\alpha}$ for

$$(\lambda x^{\alpha}\ .\ (\lambda y^{\beta}\ .\ x^{\alpha})^{(\beta\rightarrow\alpha)})^{(\alpha\rightarrow(\beta\rightarrow\alpha))}$$

We now show by examples how verbs, adjectives, and prepositions can be represented as sense typed lambda expressions.

The transitive verb *hire* could be represented by the sense typed lambda expression

$$\textit{hire} : \mathbf{Company}\rightarrow\mathbf{Person}\rightarrow(\mathbf{Company}\Rightarrow\mathbf{Person})$$

This lambda expression shows that *hire* is a function, such that, when the first argument has a noun of type subsumed by **Company** as an instance and the second argument has a noun of type subsumed by **Person** as an instance, a meaningful expression of compound type (**Company** $\Rightarrow$ **Person**) results. **Company** and **Person** are the types of the *largest* categories of nouns for which it makes sense to use the verb *hire* (the highest node in the sense type decision tree).

An instance of a meaningful verb expression is

$$\textit{hire}(\textit{Oracle}, \textit{grads}) :\ \mathbf{Company} \Rightarrow \mathbf{Person}$$

i.e., *Oracle hire grads.*

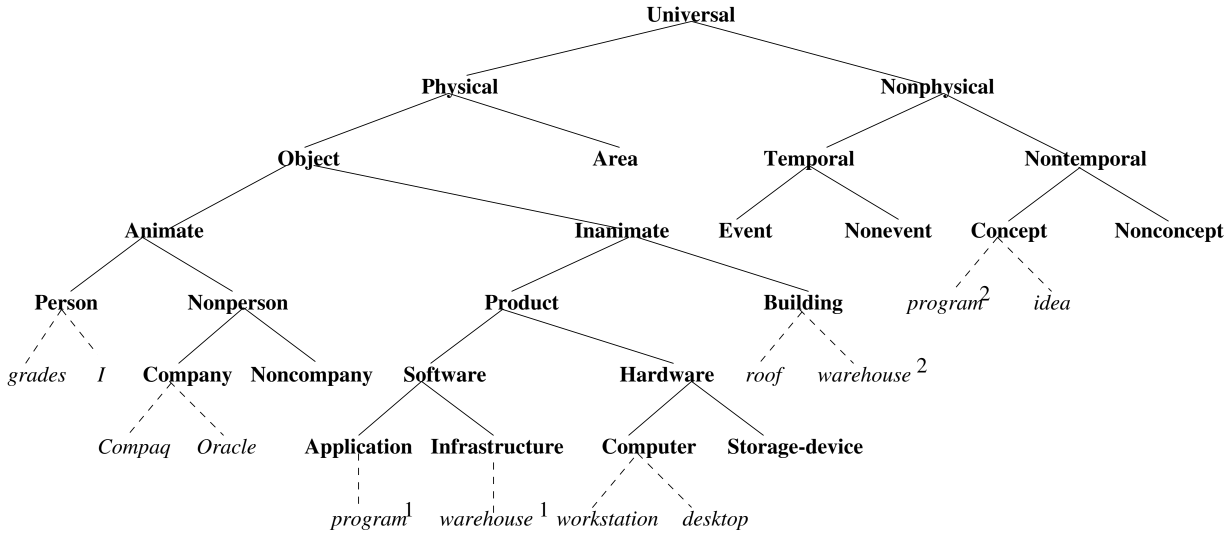

Figure 3: A small sense type decision tree with instances

We choose to use a prefix notation for functions, because this representation eliminates the variations of the locations of words in a sentence.

A meaningless verb expression such as

$$hire(Compaq, program)$$

is rejected by our system, because *program* could be of type **Application**, which is subsumed by **Inanimate**, and **Animate**||**Inanimate**.

Adjectives are represented in a similar way. The adjective *green* is represented by the sense typed lambda expression

$$green : \textbf{Physical} \rightarrow \textbf{Physical}$$

This lambda expression shows that *green* is a function of type **Physical** → **Physical**, such that, when the argument has a noun of type subsumed by **Physical** as an instance, a noun phrase of type **Physical** is produced. As a result, the following adjective sense type lambda expression

$$green(roof) : \ \textbf{Physical}$$

is meaningful, where-as the adjective expression

$$green(ideas)$$

is meaningless, because *idea* is of type **Concept** and **Concept** || **Physical**.

Conjunctions and prepositions are also represented as sense typed lambda expressions.

The conjunction *and* is represented by the sense typed lambda expression

$$(\lambda x \ y \ . \ x \ and \ y)^{'\alpha \rightarrow' \alpha \rightarrow' \alpha}$$

where the symbol $'\alpha$ is a sense type variable. The values of the variable $'\alpha$ depend on the role of the conjunction *and* in a sentence.

The preposition *with* shows the complexity of English sentences. It is represented by the sense typed lambda expression

$$(\lambda x \ y \ . \ x \ with \ y)^{'\alpha \rightarrow' \alpha \rightarrow' \alpha}$$

where the symbol $'\alpha$ is a type variable. The values of the variable $'\alpha$ depend on the sense of the preposition *with* in a sentence. We need to consider a proof system that supports the representation of a *context*, because the sense type of *with* in a sentence depends on the context in which it is used.

All the words of the lexicon are introduced by *typing axioms*. A noun or a pronoun has attached an atomic type that is a *leaf* in a tree proof, i.e., the label of the *smallest* category to which it belongs. Examples of axioms:

$$program^{\textbf{Appl}} \qquad program^{\textbf{Concept}}$$

$$hire^{\textbf{Comp} \rightarrow \textbf{Prs} \rightarrow (\textbf{Comp} \Rightarrow \textbf{Prs})}$$

$$grads^{\textbf{Prs}} \qquad Oracle^{\textbf{Comp}}$$

$$with^{'\alpha \rightarrow' \alpha \rightarrow' \alpha} \qquad and^{'\alpha \rightarrow' \alpha \rightarrow' \alpha}$$

(We abbreviated **Application** to **Appl**, **Company** to **Comp**, and **Person** to **Prs**.)

The words of the *lexicon* have sense types attached to them, in addition to the usual syntactic and semantic information. A sample lexicon is shown below.

$< programs^1$, Noun_Root{program}, Form{plural}, $program^{\mathbf{Appl}} >$

$< program^2$, Noun_Root{program}, Form{singular}, $program^{\mathbf{Con}} >$

$< green$, Adj, $green^{\mathbf{Physical} \rightarrow \mathbf{Physical}} >$

$< hired$, Verb_Root{hire}, Form{past}, $(\lambda x^{\mathbf{Comp}} y^{\mathbf{Prs}} \, . \, x \; hire \; y)^{\mathbf{Comp} \rightarrow \mathbf{Prs} \rightarrow (\mathbf{Comp} \Rightarrow \mathbf{Prs})} >$

$< developed^1$, Verb_Root{develop}, Form{past}, $(\lambda x^{\mathbf{An}} y^{\mathbf{Prod}} \, . \, x \; develop \; y)^{\mathbf{An} \rightarrow \mathbf{Prod} \rightarrow (\mathbf{An} \Rightarrow \mathbf{Prod})} >$

$< developed^2$, Verb_Root{develop}, Form{past}, $(\lambda x^{\mathbf{An}} y^{\mathbf{Con}} \, . \, x \; develop \; y)^{\mathbf{An} \rightarrow \mathbf{Con} \rightarrow (\mathbf{An} \Rightarrow \mathbf{Con})} >$

(We abbreviated **Animate** to **An**, **Concept** to **Con**, and **Product** to **Prod**.)

4 The Proof System

We follow the style used in [4] for describing typed lambda calculus and define a *strongly sense typed* language using a formalism based on logic. Specifically, we will define a proof system for deriving the sentence structures of the grammar in Figure 1 and their sense types simultaneously. We will define *axioms* to introduce the nouns and their associated atomic sense types that are in the sense type decision tree. We will also define *inference rules* to construct compound expressions and their sense types according to the grammar rules.

A sense typing axiom has the form

$$w^L$$

meaning that the noun w has type L.

One form of inference rule is

$$\frac{E_1{}^{\sigma_1}, \cdots, E_k{}^{\sigma_k}}{E^{\sigma}}$$

This rule says that if $E_1, \cdots, E_k$ are well-formed expressions of sense types $\sigma_1, \cdots, \sigma_k$, respectively, then E is a well-formed expression of type σ. The constituents above the horizontal line are called the *premises* of the inference rule and the constituent below the line the *conclusion*.

The inference rule above is not powerful enough to capture the context-sensitive syntax of expressions that declare the types of variables. To take the types of variables into account, we need typing assertions of form

$$\Gamma \triangleright E^{\tau}$$

where Γ is a type assignment of the form $\Gamma = \{x_1{}^{\sigma_1}, \cdots, x_k{}^{\sigma_k}\}$. The general form of a typing rule with contexts is

$$\frac{\Gamma_1 \triangleright E_1{}^{\sigma_1}, \cdots, \Gamma_k \triangleright E_k{}^{\sigma_k}}{\Gamma \triangleright E^{\sigma}}$$

When we use the axioms and the inference rules to derive a sentence, we obtain a *derivation tree*. The root of the derivation tree is the sentence and the leaves are instances of axioms. The *internal nodes* are conclusions of instantiated rules and they have the corresponding premises as their immediate sons. When displaying a derivation tree it is common to have the root at the bottom rather than at the top; hence the son is above the father. T1 to T3 in Figure 4 are examples of derivation trees.

The language **SNL** generated by the grammar in Figure 1 is given by axioms and inference rules. Informally, a meaningful sentence of **SNL** is a string obtained by concatenating words according to their sense types. We present below some axioms and inference rules for **SNL** without context.

1. The subsumming relation

$$\frac{L_2 \sqsubseteq^* L_1 \qquad n^{L_2}}{n^{L_1}}$$

2. Composition rules for noun phrases:

(**Det Noun**) (as in *a warehouse*):

$$\frac{d^{'\alpha \rightarrow '\alpha} \qquad n^L}{(d \; n)^L}$$

(**Det Adj Noun**) (as in *a green roof*):

$$\frac{d^{'\alpha \rightarrow '\alpha} \qquad (\lambda x \, . \, a \; x)^{L \rightarrow L} \qquad n^L}{(d \; a \; n)^L}$$

(**Noun and Noun**) (as in *Oracle and Compaq*):

$$\frac{n_1{}^L \qquad (\lambda x \; y \, . \, x \; and \; y)^{'\alpha \rightarrow '\alpha \rightarrow '\alpha} \qquad n_2{}^L}{(n_1 \; and \; n_2)^L}$$

3. Composition rules for verb phrases:

(**TrVerb NP**) (as in *to build a warehouse*):

$$\frac{(\lambda x \; y . \; x \; t_v \; y)^{L_1 \rightarrow L \rightarrow (L_1 \Rightarrow L)} \qquad n^L}{(\lambda x \, . \, x \; t_v \; n)^{L_1 \rightarrow (L_1 \Rightarrow L)}}$$

(**NP TrVerb**) (as in *Compaq developed*)

$$\frac{n^L \qquad (\lambda x \; y \, . \, x \; t_v \; y)^{L \rightarrow L_2 \rightarrow \mathbf{M}}}{(\lambda y^{L_2} \, . \, n \; t_v \; y)^{L_2 \rightarrow \mathbf{M}}}$$

T1:

$$\frac{\mathit{Oracle}^{\mathbf{Comp}} \qquad (\lambda x^{'\alpha} y^{'\alpha}\ .\ x\ \mathit{and}\ y)^{'\alpha \to '\alpha \to '\alpha} \qquad \mathit{Compaq}^{\mathbf{Comp}}}{(\mathit{Oracle\ and\ Compaq})^{\mathbf{Comp}}}$$

using rule (**Noun** *and* **Noun**)

T2:

$$\frac{(\mathit{Oracle\ and\ Compaq})^{\mathbf{Comp}} \qquad (\lambda x^{\mathbf{Comp}} y^{\mathbf{Prod}}\ .\ x\ \mathit{develop}\ y)^{\mathbf{Comp} \to \mathbf{Prod} \to (\mathbf{Comp} \Rightarrow \mathbf{Prod})}}{(\lambda y^{\mathbf{Prod}}\ .\ (\mathit{Oracle\ and\ Compaq})\ \mathit{develop}\ y)^{\mathbf{Prod} \to (\mathbf{Comp} \Rightarrow \mathbf{Prod})}}$$

using rule (**NP TrVerb**) and *verb specialization* $\mathbf{Comp} \sqsubseteq *\ \mathbf{An}$

T3:

$$\frac{\dfrac{(\mathit{Oracle\ and\ Compaq})^{\mathbf{Comp}} \qquad (\lambda x^{\mathbf{An}} y^{\mathbf{Prod}}\ .\ x\ \mathit{develop}\ y)^{\mathbf{An} \to \mathbf{Prod} \to (\mathbf{Comp} \Rightarrow \mathbf{Prod})}}{(\lambda y^{\mathbf{Appl}}\ .\ (\mathit{Oracle\ and\ Compaq})\ \mathit{develop}\ y)^{\mathbf{Appl} \to (\mathbf{Comp} \Rightarrow \mathbf{Appl})}} \qquad \mathit{program}^{\mathbf{Appl}}}{((\mathit{Oracle\ and\ Compaq})\ \mathit{develop\ program})^{(\mathbf{Comp} \Rightarrow \mathbf{Appl})}}$$

using $T2$ *and rule* (**TrVerb NP**) *because* $\mathbf{Appl} \sqsubseteq \mathbf{Prod}$

Figure 4: Derivation trees

4. Verb specialization

$$\frac{L_3 \sqsubseteq^* L_1 \qquad v^{L_1 \to L_2 \to (L_1 \Rightarrow L_2)}}{v^{L_3 \to L_2 \to (L_3 \Rightarrow L_2)}}$$

$$\frac{L_3 \sqsubseteq^* L_2 \qquad v^{L_1 \to L_2 \to (L_1 \Rightarrow L_2)}}{v^{L_1 \to L_3 \to (L_1 \Rightarrow L_3)}}$$

We present in Figure 4 some instances of derivation trees using some of the inference rules defined above.

5 Summary and Future Work

The construction of a sense type decision tree of atomic semantic categories and the view of English text as a strongly-typed language lead to compound sense types, which are compound semantic categories. This approach provides a new way of sentence categorization, which increases the precision of document exploration.

Many applications of the approach developed in this paper are in progress. Some of our work focuses on automatic translation of compound categories from one language to another. Other work is focused on the query representation for handling compound categories.

References

[1] M.Z. Bleyberg, C. Sexton, and B. Hankley. Sense typed lambda-calculus for natural language processing. Technical Report 96-16, Computer and Information Sciences, Kansas State University, 1996.

[2] R. Feldman and I. Dagan. Knowledge discovery in textual databases. In *KDD'95 - Proc. 1st Intl. Conf. on Knowledge Discovery and Data Mining*, 1995.

[3] U. Hahn and K. Schnattinger. Knowledge mining from textual sources. In *CIKM'97 - Proc. 6th Intl. Conf. on Information and Knowledge Management*, pages 83–90, 1997.

[4] John Mitchell. *Foundations for Programming Languages.* The MIT Press, 1996.

[5] Michael Moortgat. Categorial type logics. In *Handbook of Logic and Language*, 1998.

[6] T. Pedersen and R. Bruce. Knowledge lean word-sense disambiguation. In *Proc. 15th Nat. Conf. on Artificial Intelligence*, 1998.

[7] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993.

[8] Clark Sexton. *Understanding English Texts.* PhD thesis, Computer and Information Sciences, Kansas State University, 1998.

[9] Fred Sommers. Types and ontology. *Philosophy Review*, (72), July 1963.

[10] Raymond Turner. Types. In *Handbook of Logic and Language*, 1998.