Technical Report CIS 99-5
INFERENCE RULES FOR TEXT DATA MINING

Maria Zamfir Bleyberg
Computing and Information Sciences Department
Kansas State University
Manhattan, KS, 66506

Phone: 785-532-6350
e-mail: maria@cis.ksu.edu

Abstract

In this paper, we propose a new alternative theory for deep knowledge discovery from natural language documents. We treat natural language documents as we treat strongly-typed functional programming languages and introduce *categories* as types. We define sentences and their categories simultaneously using category *axioms* and *inference rules*. *Primitive* categories are nodes of a *sense* type decision tree. The inference rules capture the *meaningful* relationships among primitive categories. Axioms and inference rules are used to construct tree proofs, which create *structured* categories, validate category hypotheses, and eliminate ambiguities. Structured categories increase the precision of document exploration.

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

1 Introduction

In processing of computing languages, the concept and role of types are well established. For such artificial languages, there are commonly accepted *primitive* types and constructors for *structured* types. Sentences of the language are parsed according to well-defined rules of grammar and type consistency.

Many Knowledge Discovery in Texts (KDT) paradigms are based on simple forms of text categorization [6, 3]. Text categorization is the assignment of one or more labels from a pre-defined set to a document. Progress has been made by imposing simple structures on the pre-defined set of labels, such as, the keyword graphs and background knowledge [5, 7, 4, 8]. In categorization, the assignment of the labels to words assumes the independence of the words.

In this paper, we show that English text categorization can be improved by introducing *structured* categories as types associated to its constructs. *Primitive* categories are introduced as axioms and the relationships among categories are introduced as inference rules. Primitive categories are also nodes of a sense type decision tree. Axioms and inference rules are used to construct tree proofs, which create *structured* categories and validate category hypotheses. The tree proofs also help solve subtle *ambiguities* that occur when a noun may have many categories associated to it. Structured categories increase the precision of document exploration. For example, let us consider the following sentences:

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

Figure 1: English text displaying ambiguities

A simple categorization of the words of the sentences [a] and [b] would not separate the different meaning they have, even in the presence of background knowledge. Each of these sentences has a lexical ambiguity. In our approach, assuming that we have the primitive categories **Company**, **Person**, **Building**, and **Software** with the subcategory **Infrastructure** (see Figure 3), sentences [a] and [b] get the following categorization:

$$[a]: \ \mathbf{Company} \Rightarrow (\mathbf{Person} \Rightarrow \mathbf{Building})$$

$$[b]: \ \mathbf{Company} \Rightarrow (\mathbf{Person} \Rightarrow \mathbf{Infrastructure})$$

The present work is based upon the work reported in [2], in which a *strongly sense typed* subset of English has been defined following the style used in [9] for describing typed lambda calculus.

2 Sense Type Decision Trees

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

This grammar gives us firm intuitions about the structure of a sentence, but it does not distinguish among the language's meaningful sentences.

According to Sommers' sense logic [11], there are *two* types of sentences: *meaningful* ones and *meaningless* ones. 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 properties (adjectives and verbs of the lexicon) are used to partition the set of nouns at a node into *complimentary* sets. An

S	::=	**NP VP**
NP	::=	**PrName** | **Noun** | **NP Conj NP** | **Det Noun** | **Det Adj Noun** | **S**
VP	::=	**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(ed)* | *develop(ed)* | *join(t)* | *build(ing)*
Det	::=	*a* | *the*
Adj	::=	*joint* | *training* | *green*
Prep	::=	*to* | *with* | *for*
Conj	::=	*and*

Figure 2: A simple *context-free* grammar

example of a binary sense type decision tree (with some instances) is given in Figure 3. 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.

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*: *a workstation runs.*

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 is intended to partition the nouns and pronouns 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. An interactive tool for the construction of such a binary sense type decision tree has been presented in [10].

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 binary tree structure to $\mathcal{P}(U)$. We call a label of a class, $L_{i_1 \cdots i_k 1}$, a *primitive* type, i.e., a *primitive* category. The partial order $\subseteq$ on $\mathcal{P}(U)$ induces a partial order $\sqsubseteq$ on the collection $\mathcal{L}$ of the primitive types of the $\mathcal{P}(U)$ classes. The following sequence is an example of a path in the binary hierarchy:

$$L_1 \sqsubseteq L_{11} \sqsubseteq \cdots \sqsubseteq L_{i_1 \cdots i_k} \sqsubseteq L_{i_1 \cdots i_k 1}, \cdots \tag{1}$$

We call *generalization* moving up in the hierarchy and *specialization* moving down in the hierarchy.

Let L_i, L_j be arbitrary 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. We say that L_i and L_j are *comparable* and that L_j

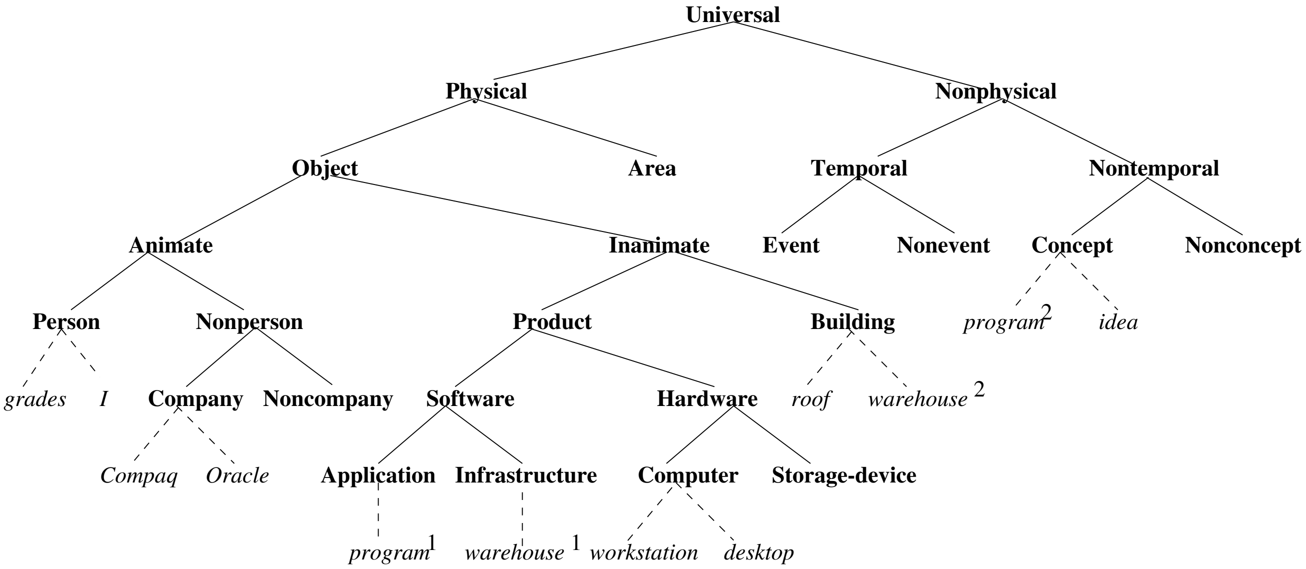

Figure 3: A small sense type decision tree with instances

subsumes L_i. For example, we have **Person** $\sqsubseteq^*$ **Animate**, i.e., **Animate** 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*. For example, we have **Software** $\parallel$ **Hardware**.

The *sense type expressions* that we consider in this work are given by the following definition.

Given the set $\mathcal{L}$ of primitive types, we define types as follows:

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

Each primitive type $L \in \mathcal{L}$ denotes the class $[U]$ of nouns it represents. Each type $(\alpha \Rightarrow \beta)$, called a *structured* type, 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 structured type is the collection of *meaningful* sentences. A sentence is *meaningless* if no type can be assigned to it.

The interpretation of a structured type expression is based on the concept of a *typed lambda abstraction*, which will be presented in the next section.

3 A Sample Lexical Database with Categories

Lambda abstraction [9] 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.

In general, we write $(\lambda x\ y\ .\ x)^{\alpha\to\beta\to\alpha}$ for $(\lambda x^{\alpha}\ .\ (\lambda y^{\beta}\ .\ x^{\alpha})^{(\beta\to\alpha)})^{(\alpha\to(\beta\to\alpha))}$.

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

$$hire : \textbf{Company} \to \textbf{Person} \to (\textbf{Company} \Rightarrow \textbf{Person})$$

This 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 structured type (**Company** ⇒ **Person**) results. **Company** and **Person** are the types of the *largest* groups of nouns for which it makes sense to use the verb *hire* (the highest node in the decision tree). (In [1], verbs are represented as untyped lambda expressions serving a different purpose.)

An instance of a meaningful verb expression of structured type **Company** ⇒ **Person** is $hire(Oracle, grads)$: **Company** ⇒ **Person**, or,
$(Oracle\ hire\ grads)$: **Company** ⇒ **Person**.

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

$$hire(Compaq, program^1)$$

is rejected by our system, because $program^1$ is 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

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

This lamda 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 lambada expression

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

is meaningful, where-as the adjective expression

$$green(ideas)$$

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

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

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

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

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

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

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 formalism 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.

Pronoun *I* is of type **Person**, pronoun *it* is of type **Universal** and determiner *the* is of type **Universal**.

All the nouns of the lexicon are introduced by *typing axioms*. A noun or a pronoun has attached a type that is a *leaf* in the sense type decision tree, i.e., the label of the *smallest* class to which it belongs. Examples of axioms:

$$program^{\mathbf{Appl}} \qquad program^{\mathbf{Con}} \qquad hire^{\mathbf{Comp} \to \mathbf{Person} \to (\mathbf{Comp} \Rightarrow \mathbf{Person})}$$

$$grads^{\mathbf{Person}} \qquad Oracle^{\mathbf{Comp}} \qquad with^{'\alpha \to '\alpha \to '\alpha} \qquad and^{'\alpha \to '\alpha \to '\alpha}$$

(We abbreviated **Application** to **Appl**, **Company** to **Comp**, **Concept** to **Con**).

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, $green^{\mathbf{Physical} \to \mathbf{Physical}}$ $>$
$<$ *hired*, Verb_Root{hire}, Form{past},
$(\lambda x^{\mathbf{Comp}} y^{\mathbf{Person}}\ .\ x\ hire\ y)^{\mathbf{Comp} \to \mathbf{Person} \to (\mathbf{Comp} \Rightarrow \mathbf{Person})}$ $>$
$<$ *build*[1], Verb_Root{build}, Form{inf},
$(\lambda x^{\mathbf{Anm}} y^{\mathbf{Building}}\ .\ x\ build\ y)^{\mathbf{Anm} \to \mathbf{Building} \to (\mathbf{Anm} \Rightarrow \mathbf{Building})}$ $>$
$<$ *build*[2], Verb_Root{build}, Form{inf},
$(\lambda x^{\mathbf{Anm}} y^{\mathbf{Infrast}}\ .\ x\ build\ y)^{\mathbf{Anm} \to \mathbf{Infrast} \to (\mathbf{Anm} \Rightarrow \mathbf{Infrast})}$ $>$
$<$ *developed*[1], Verb_Root{develop}, Form{past},
$(\lambda x^{\mathbf{Anm}} y^{\mathbf{Prod}}\ .\ x\ develop\ y)^{\mathbf{Anm} \to \mathbf{Prod} \to (\mathbf{Anm} \Rightarrow \mathbf{Prod})}$ $>$
$<$ *developed*[2], Verb_Root{develop}, Form{past},
$(\lambda x^{\mathbf{Anm}} y^{\mathbf{Con}}\ .\ x\ develop\ y)^{\mathbf{Anm} \to \mathbf{Con} \to (\mathbf{Anm} \Rightarrow \mathbf{Con})}$ $>$

(We abbreviated **Infrastructure** to **Infrast** and **Animate** to **Anm**.)

4 Structured Categories

We follow the style used in [9] for describing typed lambda calculus and define a *strongly sense typed* language using a formalism based on logic. Specifically, we will introduce the sentence structures of the grammar in Figure 2 and their sense types simultaneously using *axioms* and *inference rules*.

A sense typing axiom has the form

$$w^L$$

meaning that the word w has type L.

The compound expressions and their sense types are defined using inference rules. 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 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 T6, which are shown at the end of this section, are examples of derivation trees.

The language **SNL** generated by the grammar in Figure 2 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 covers the generalization of nouns:

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

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*)

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

3. Adjective specialization

$$\frac{L_2 \sqsubseteq^* L_1 \qquad a^{L_1 \to L_1}}{a^{L_2 \to L_2}}$$

4. Composition rules for nouns involving the conjunction *and*:

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

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

5. Composition rules for nouns involving the preposition *with*:

(**NP** *with* **NP**)$_{component}$
(as in *a warehouse with a roof*)

$$\frac{{n_1}^L \qquad (\lambda x\ y\ .\ x\ with\ y)^{'\alpha \to '\alpha \to '\alpha} \qquad {n_2}^L}{(n_1\ with\ n_2)^L}$$

(**Noun** *with* **Noun**)$_{together}$
(as in *spaghetti with meatballs*)

$$\frac{{n_1}^L \qquad (\lambda x\ y\ .\ x\ with\ y)^{'\alpha \to '\alpha \to '\alpha} \qquad {n_2}^L}{(n_1\ with\ n_2)^L}$$

6. Composition rules for verb phrases:

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

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

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

$$\frac{n^L \qquad (\lambda x\ y\ .\ x\ t_v\ y)^{L \to L_2 \to (L \Rightarrow L_2)}}{(\lambda y^{L_2}\ .\ n\ t_v\ y)^{L_2 \to (L \Rightarrow L_2)}}$$

7. 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 below some instances of derivation trees using the inference rules defined above, which can also be viewed as steps for the categorization of the sentences in Figure 1.

(T1)

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

$using\ rule\ 4:$ (**Noun** *and* **Noun**)

(T2)

$$\frac{a^{'\alpha \to '\alpha} \qquad (\lambda x\ .\ green\ x)^{\mathbf{Building} \to \mathbf{Building}} \qquad roof^{\mathbf{Building}}}{(a\ green\ roof)^{\mathbf{Building}}}$$

$using\ rule\ 2:$ (**Det Adj Noun**) *and rule* $3:$ *adjective specialization*

(T3)

$$\frac{warehouse^{\mathbf{Building}} \qquad (\lambda x^{'\alpha} y^{'\alpha}\ .\ x\ with\ y)^{'\alpha \to '\alpha \to '\alpha} \qquad (a\ green\ roof)^{\mathbf{Building}}}{(warehouse\ with\ a\ green\ roof)^{\mathbf{Building}}}$$

using rule 5: (**NP** *with* **NP**)$_{component}$

(T4)

$$\frac{warehouse^{\mathbf{Building}} \qquad (\lambda x^{'\alpha} y^{'\alpha}\ .\ x\ with\ y)^{'\alpha \to '\alpha \to '\alpha} \qquad Oracle^{\mathbf{Comp}}}{(warehouse^{\mathbf{Building}}\ with\ Oracle^{\mathbf{Comp}})^{??}}$$

rule (**NP** *with* **NP**)$_{component}$ *is not allowed, because* $\mathbf{Comp} \sqsubseteq^* \mathbf{Animate}$, $\mathbf{Building} \sqsubseteq^* \mathbf{Inanimate}$ *and* $\mathbf{Animate} \| \mathbf{Inanimate}$

(T5)

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

$using\ rule\ 6:$ (**NP TrVerb**) and rule 7: *verb specialization* $\mathbf{Comp} \sqsubseteq *\ \mathbf{Anm}$

(T6)

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

$using\ T5\ and\ rule\ 6:$ (**TrVerb NP**) *because* $\mathbf{Appl} \sqsubseteq *\ \mathbf{Product}$

From the instances presented above it is clear that proof trees properly assign the appropriate structured categories to sentences and solve many forms of ambiguity.

5 Summary and Future Work

The view of English text as a strongly-typed language leads to the concept of structured categories. This approach represents 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 structured categories from one language to another. Other work is focused on the query representation for handling structured categories.

References

[1] James Allen. *Natural Language Understanding.* Benjamin Cummings, 1995.

[2] 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.

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

[4] R. Feldman and H. Hirsh. Mining associations in text in the presence of background knowledge. In *KDD'96 - Proc. 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, 1996.

[5] U. Hahn and K. Schnattinger. Deep knowledge discovery from natural language texts. In *KDD'97 - Proc. 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 175–178, 1997.

[6] 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.

[7] M. Hearst and C. Karadi. Searching and browsing text collections with large category hierarchies. In *Proc. of the ACM SIGCHI Conf. on Human factors in Computing Systems (CHI)*, 1997.

[8] M. Hearst and H. Schutze. Customizing a lexicon to better suit a computational task. In *Proc. of the ACL SIGLEX Workshop, Columbus, Ohio*, 1993.

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

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

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