Computing an Ontological Semantics for a Natural Language Fragment

Szymczak, Bartlomiej Antoni; Nilsson, Jørgen Fischer; Jensen, Per Anker

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Computing an Ontological Semantics for a Natural Language Fragment

Bartłomiej Antoni Szymczak

DTU Informatics
Technical University of Denmark

International Language Studies and Computational Linguistics
Copenhagen Business School

Kongens Lyngby 2010
IMM-PHD-2010-242

Abstract

The key objective of the research that has been carried out has been to establish theoretically sound connections between the following two areas:

- Computational processing of texts in natural language by means of logical methods
- Theories and methods for engineering of formal ontologies

We have tried to establish a domain independent "ontological semantics" for relevant fragments of natural language. The purpose of this research is to develop methods and systems for taking advantage of formal ontologies for the purpose of extracting the meaning contents of texts. This functionality is desirable e.g. for future content–based search systems in contrast to today's keyword based search systems (viz., Google) which rely chiefly on recognition of stated keywords in the targeted text.

Logical methods were introduced into semantic theories for natural language already during the 60's in what is today known as Montague semantics. However, this well–established tradition addresses mainly the domain independent logical structures of language such as quantifiers/determiners by means of logic [18], such as type theory [2]. By contrast this project focuses on the domain–specific parts of language (nouns, verbs, adjectives) introducing formal so–called generative ontologies as semantic target domains for noun– and verb phrases. Such a logico–semantic theory links the meaning of a sentence phrases to nodes in the chosen ontology for the domain.

Resumé

Den centrale målsætning for den forskning, der er blevet udført har været at etablere teoretisk velfunderet forbindelse mellem følgende to områder:

- Maskinel behandling af tekster i naturlige sprog ved hjælp af logiske metoder.
- Teorier og metoder til projektering af formelle ontologier.

Vi har forsøgt at etablere en domæneuafhængig "ontologisk semantik" for relevante fragmenter af naturligt sprog. Formålet med denne forskning har været at udvikle metoder og systemer til at drage fordel af formelle ontologier med henblik på udvinding af betydningsindholdet af tekster. Denne funktionalitet er ønskelig f.eks for fremtidige indholdbaserede søgesystemer i modsætning til dagens søgeordsbaserede søgesystemer (f.eks, Google), som er afhængige først og fremmest af genkendelse af anførte nøgleord i de målrettede tekster.

Logisk metoder blev indført i semantiske teorier for naturligt sprog allerede i løbet af de 60erne i, hvad der i dag kendes som Montague semantik. Men denne veletablerede tradition adresserer primært domæneuafhængige logiske strukturer af sprog såsom kvantorer/artikler ved hjælp af logisk typeteori [2]. Derimod fokuserer dette projekt på den domænespecifikke del af sproget (substantiver, verber, adjektiver) ved at indføre formelle såkaldte generative ontologier som semantiske måldomæner for navne- og verbal fraser. Sådan en logico–semantisk teori knytter betydningen af en sætning fraser til knudepunkter i den valgte ontologi for domænet.

Acknowledgements

The author would like to thank his supervisor, prof. Jørgen Fischer Nilsson and co–supervisor, prof. Per Anker Jensen for their help and guidance.

The author gratefully acknowledges the financial support with grants from DTU Informatics, Technical University of Denmark and International Language Studies and Computational Linguistics, Copenhagen Business School.

Contents

List of Figures

Chapter 1

Motivation

1.1 Goals

Many future generation software systems offering common public and specialized services are going to apply computational representation and processing of the meaning content of text and speech.

Representing such a semantics is a task that has been undertaken repeatedly by linguists, computational linguists, philosophers, logicians, statisticians, mathematicians, computer scientists and software engineers. In addition to logical and symbolic representations, meaning of language can be represented geometrically with the help of vectors, or even physically with quantum states and quantum mechanics. Such an approach is presented in [48].

However, rather than pursuing a geometric or non–symbolic approach, in this work we have decided to explore and use a logico–algebraic approach. This is due to two main reasons. First of all, we would like to be able to describe semantics of large complex objects, such as phrases and sentences. This requires the compositionality for semantics, where meaning of complex phrases can be constructed from meanings of smaller constituent phrases. This is relatively easily accomplished with symbolic and logic approach. The second reason is that, we will use the so–called generative ontology to a large extent, and it is naturally represented in symbolic formalism. Also, we strive to achieve full formalization and implementability of our methods, which is in general hard with non–symbolic representations such as Conceptual Spaces [19].

A well-established and prominent theory for coping with natural language

meaning content is the type-logical framework devised in the Montague tradition. However, this logical tradition tends to focus on closed word classes such as determiners in order to account for the logical aspects of meaning formation independent of context [13]. This is at the expense of open classes such as adjectives, nouns, and verbs which, by contrast, intuitively play the key role in carrying meaning and the ascription of meaning with respect to a domain of discourse and a lexicon.

The word "Ontology" appeared thousands of years ago, where ancient philosophers tried to explain the essence of being. In such context, we use the capitalized word "Ontology". However, the current work deals with formal ontologies, or simply ontologies, written with lower–case "o". Formal ontology can be understood as specification of a conceptualization. It is crucial that the ontology is specified in a formal way, so that it is machine–readable and understandable. A conceptualization can be thought of as a view of the world shared by some people. Of course none of the ontologies can describe a full conceptualization of the whole world. In most cases an ontology attempts to describe only a selected part of a specific domain. [17]

This work addresses the contextual and lexical meaning contributions and compositions via the notion of formal ontologies. The latter have recently become a research focal point in domain modeling and knowledge engineering. Specifically, we try to develop theories and methodologies which bridge formal ontologies and computational linguistics in a formal logical framework for "ontological semantics".

We would like to list some assumptions/requirements of this work:

- Formal logical methods form the basis for this work. This is in contrast to the fact that in natural language analysis and processing focus is commonly put on statistical methods. We do not use any statistics.
- The project concentrates on written language only. Furthermore, we narrow our attention to English texts in scientific domains.
- The project is intended to deal with search–oriented semantics for natural language. The construction of an actual search engine is beyond the scope of this study.
- We would like to achieve semantics that goes beyond keyword–seach, yet is not as rich as we usually tend to think when talking about natural language semantics. This can be visually expressed as:

$$\text{keywords} < \text{ontological semantics} < \text{natural language semantics}$$

- The goal of the project is to investigate some possibilities of merging natural language semantics with formal ontologies. Hence, rather than providing one complete solution for doing it, the project presents various attempts, using different formalisms, languages and methods.

1.2 Partners

This work is carried out as part of a PhD-project which was carried out in association with the SIABO project. SIABO addresses methods of engineering and use of biomedical ontologies for content-based text search. From SIABO website:

> *The approach used in SIABO introduces the notion of generative ontologies, that is, ontologies providing ever more specialized concepts reflecting the phrase structure of natural language. The project seeks to set up a novel so–called "ontological semantics" mapping noun phrases into points in the generative ontology. This enables an advanced form of data mining of texts identifying paraphrases and conceptual relationships, and measuring distances between key concepts in texts. Thus, the project is unique in its attempt to provide a formal underpinning to conceptual similarity or relatedness of meaning. The project focuses on ontological engineering of biomedical ontologies applying the notion of lattices and relation-algebras, which facilitates visualization of concepts as "ontoscapes". The project has clear affinities to contemporary research in the semantic web area, to description logic as well as XML approaches and gains its distinct innovative scientific profile by means of the above mentioned notions.*

SIABO is supported by the Danish NABIIT program. SIABO partners include:

- DTU Informatics
- Group of Computational Linguistics at Copenhagen Business School
- Programming, Logic and Intelligent Systems research group at Roskilde University
- NovoNordisk A/S Research Division, Copenhagen.

1.3 More formal definitions

Throughout the text we will use First Order Predicate Logic as the framework metalanguage, in which other formalisms will be formally presented. We have chosen to do this mainly for the following reasons:

- People are usually familiar with First Order Predicate Logic, so that it becomes a convenient formal inter–field language, just as English is a convenient natural international language.
- The definitions become formal, as opposed to natural language descriptions in some sources.
- The implementation is usually presented in Prolog, which comes close to First Order Predicate Logic. This diminishes the distance between the theoretical parts and the implementation.

1.4 Semantics of the keyword–based search and its deficiencies

The big Internet boom of 1990s was followed by a huge increase in search engine research. Many successful companies emerged, most notably Google, whose search engine [11] has revolutionized the Internet.

1.4.1 Definition of keyword–based search

All of the current search engines in wide use are keyword–based. Informally, this means that the query is a collection of words, and the search returns a collection of documents that contain (usually) all of the queried words. Formally we can describe the behavior of keyword search using first order predicate logic[1] as follows.

We can represent an empty list using the constant *nil*. A non–empty list will be constructed using binary functor l, whose first argument is a head of the list, and second argument is the tail of the list, being itself a list.

We assume that the documents to be searched are stored in the *document* factual clause, which holds a list of keywords. A sample database of documents D could look as follows:

[1] First order predicate logic is described in Section 2.1.

$$document(l(insulin, l(forces, l(storage, nil)))) \tag{1.1}$$

The following set of formulae describes how a keyword–based search engine works:

$$\begin{aligned} &\forall D, Q(document(D) \wedge \\ &\quad \forall K(member(K, Q) \rightarrow member(K, D)) \\ &\quad \rightarrow query(Q, D)) \end{aligned} \tag{1.2}$$

$$\forall X, L(member(X, l(X, L))) \tag{1.3}$$

$$\forall X, Y, L(member(X, L) \rightarrow member(X, l(Y, L))) \tag{1.4}$$

As the set of formulae 1.2 – 1.4 show, querying with a list of keywords returns a document only if it contains all the keywords found in the query.

Some engines differ in behavior in one or more aspects:

- Returning documents, which contain only subset of the query keywords.
- Returning documents, which contain keywords related to the query keywords, e.g. $forced$ instead of $forces$.

Let us call the set of formulae 1.2 – 1.4 $KBSE$. Coupled together with our sample database D from formula 1.1, we can make the following sample inference:

$$\begin{aligned} KBSE \cup D \quad \models \quad & query(l(forces, l(storage, nil)), \\ & l(insulin, l(forces, l(storage, nil)))) \end{aligned} \tag{1.5}$$

1.4.2 Problems with keyword–based search

The keyword querying has some advantages. One of them is simplicity of implementation. The search engine presented in formulae 1.2 – 1.4 can be implemented in Prolog in a few lines. Unfortunately, such a primitive approach to information retrieval results in a number of disadvantages. Some problems stem from the fact that the words in the document and the ones in the query could be different, while we would still like to have a match. This can be overcomed to some extent lemmatization and synonym detection. However, a number of other problems is present, which are enumerated below with accompanying examples.

Scattered keywords

Let us consider the following query: `insulin causes storage`. The user is interested in the concept of insulin forcing a storage of something.

Most search engines will return an article, in which `insulin` and `storage` are present in one sentence, and `causes` is present in another sentence. This behavior follows from the fact that the search engine does not recognize concepts appearing in the text, but rather relies on presence of words in the whole document.

Multiple queries

Most search engines have a feature which allows the user to search for a sequence of keywords, which must appear contiguously in the returned document. Usually, such a sequence must be enclosed in double quotes in the query. This allows the user to overcome the problem of scattered keywords to a certain extent. One could, e.g. search for `‘‘insulin causes storage’’`, so that documents with those keywords scattered around will not be shown in the result.

Unfortunately, the user is usually forced to pose multiple queries, as the document author might have written the text in a different way. The author could use different wording, phrased the document in a different manner, etc.

As an example, the user might be forced to query with the following sequences of keywords:

- `‘‘insulin causes storage’’`
- `‘‘insulin is causing storage’’`
- `‘‘storage caused by insulin’’`
- ...

For each query different articles will be returned (if any).

An ontological search engine should realize that all of those sequences of keywords are paraphrases. It should therefore be necessary to simply fire one query only. It is a key goal in our work to achieve such a functionality.

Lack of underlying ontology

Without an underlying ontology, the search engine will not be able to realize that “forcing” is in fact a kind of “causing”. The user therefore needs to fire up additional queries:

- ‘‘insulin forces storage’’
- ‘‘insulin is forcing storage’’
- ‘‘storage forced by insulin’’
- ...

Language dependency

The deficiencies of keyword–based searching are due to the fact that the engine compares text, not the semantics. If the engine was able to extract semantics from the text and from the query and compare them, instead of comparing the text itself, many problems could be easily overcome.

Comparing the ontological concepts instead of the words that describe them could allow new ways of information retrieval, e.g. one could query in English, but get an article in Danish as a result.

1.5 Comprehensive natural language semantics

At one end of the spectrum we have the simple keyword functionality presented in the previous section. At the other far end of the spectrum there is a very rich semantics, where the meaning of a sentence is presented as a logical formula in an expressive (usually undecidable) logic formalism. A common choice is the Type Theory comprising simply–typed λ–calculus. [3] A very elaborate linguistic system based on it is presented in [46]. However, we focus on using ontologies as basis for semantic domain, and we present one approach merging generative ontologies and type theory in Section 5.12 on page 98.

In this “rich” scenario the meaning of a sentece or a phrase is captured as a logical formula, which facilitates the use of logical consequence as the means of establishing how the meaning of two sentences relate. There are, however, many problems with using such a semantics for search purposes.

Formalisms used in this scenario (e.g. First Order Predicate Logic, λ–calculus) use variables extensively, and therefore they focus a lot on quantifiers as meaning–carying elements (corresponding to determiners like “every”, “all”, “some”). They also put stress on logical connectives/operators. However, scientific texts in biomedical domain do not tend to convey meaning with quantifiers and connectives. The meaning in these texts is concentrated around nouns, verbs, adjectives and prepositions. Even if we find rare cases of quantifier–related

meaning in sentences, we claim that it is not important (maybe even undesired) to make use of that meaning for the purpose of search. Consider the sentence "All beta cells secrete insulin". This could be translated to First Order Predicate Logic as $\forall X(betacell(X) \rightarrow secrete(X, insulin))$. Now consider "Many beta cells secrete insulin". This would translate to the formula $\exists X(betacell(X) \wedge secrete(X, insulin))$. We can observe how a lot of focus has been put on quantification and operators. This seems irrelevant from the search–perspective, as it is hard to imagine a user query where one of these sentences should be returned as the result and the other should not. Also, deciding (even non–automatically, i.e. by human) how to describe a meaning of a sentence in First Order Predicate Logic or Type Theory is very difficult. The textbooks are full of perfect examples, which seem to follow the building blocks of a given logic, so they can be translated easily. Unfortunately, it is not so easy with real life sentences, let alone in an automatic manner.

The formalisms commonly used for representing natural language semantics come with logical consequence, which can in principle be used for search. The search engine can return all documents containing sentences such that their meaning entail the query. However, this approach suffers from multiple problems. First Order Predicate Logic and Type Theory are undecidable formalisms, which means that sometimes we might not be able to conclude whether a given sentence should be included in the search results. Another problem is that an inconsistent sentence may appear in all search results (as anything follows from it). Also, basing search on logical consequence seems to be a bad choice for search purposes, because it does not mirror the intentions of the user. Consider two sentences: "All beta cells secrete insulin" and "Not all beta cells secrete insulin". Clearly, neither of these follow from the other (as a matter of fact they are opposites). However, in search context, if one of them matches a query, we would like the other one to match as well. This is because while searching we are mostly concerned with the "aboutness" of a sentence (what a sentence is about) rather than the propositional content that focuses on variables, quantifiers, and logical connectives.

In the current work, we try to find a better semantic domain, which will be suitable for search. We will base it on a notion of generative ontologies, and we will investigate the computational aspect of it. We would like to capture more meaning than the keyword semantics, but less than the rich logic–based semantics. As a matter of fact, the keyword–semantics will be our fallback mechanism, i.e. any phrase that is not understood well by our methods will produce corresponding keywords as result.

1.6 Ontology–enabled browsing search

Basing the search semantics on a generative ontology has a big advantage, i.e. it enables search by browsing the indexed generative ontology.

Imagine that when processing the to–be–searched documents for every ontological concept found we create a link from the generative ontology to that document. We can now browse the generative ontology, which for every node would display the number of documents indexed under that node. It could obviously also list the links to the documents (or even sentences) that contain this node in their semantics. Example:

Let us say we want to find some documents about inhibiting the transport of glucose. We start browsing the ontology at "inhibition". The system says that there is 10^6 documents containing this concept. It also displays the most general subclasses, e.g. "inhibition of substance", "inhibition of process", etc. We select the latter, and we can see that there is 10^5 documents about "inhibition of process". This is still too many, so we browse down different processes, and we get to "inhibition of transport", which has 10^4 documents. Again, we narrow this concept down to "inhibition of (transport of substance)", which has 10^3 documents. We browse further down, so that we narrow our concept to "inhibition of (transport of glucose)". At this point the system says there is only 10^2 documents indexed. We could continue browsing down to even more specific concepts (e.g. if we wanted the inhibition to happen in liver), but we decide that we want to see the list of the documents.

This way of finding documents has some advantages compared to keyword querying. The most important thing to notice is that the query is a concept in the ontology, hence the query is not expressed in natural language. The same concept would correspond to a series of different noun phrases, verb phrases, sentences, or even languages.

Such a browsing is of course not free of drawbacks. The user needs to be a domain expert in order to browse the ontology efficiently.

We believe that the browsing search approach could become a useful tool in biomedical domains, where large ontologies exist, and people are fairly familiar with them.

1.7 Importance of fast information retrieval

In todays world, people increasingly depend on fast access to information. We have entered the era of "information society", where Internet search engines are

the primary tool for each computer user.

Subsequently, any improvement in the way people search for information constitutes a big leap forward for computer users. People "google" 3 billion times a day[30]. Simple calculation shows that if by improving information retrieval techniques we could save on average one second of person's time per search, this amounts to:

$$\begin{aligned}
& (\text{timesavedpersearch}) \cdot (\text{numberofsearchesperday}) \\
= \;& 1\text{s} \cdot 3 \cdot 10^9 \\
= \;& 3 \cdot 10^9\text{s} \\
= \;& 3 \cdot 10^9\text{s} \cdot \frac{1\text{min}}{60\text{s}} \\
= \;& 5 \cdot 10^7\text{min} \cdot \frac{1\text{h}}{60\text{min}} \\
= \;& 8.(3) \cdot 10^5\text{h} \cdot \frac{1\text{d}}{24\text{h}} \\
\approx \;& 34722\text{d} \cdot \frac{1\text{y}}{365\text{d}} \\
\approx \;& 95\text{y}
\end{aligned}$$

Thus, each day, we could save a century of humanity's time (95 man–years) by one–second–per–person improvement. [2]

In section 1.4 we explain why most people unnecessarily waste not one second, but much more time on machine–aided searching using today's state–of–the–art technology.

Our hope is that by merging natural language semantics with formal ontologies we could achieve semantic representation of text that could be used for more intelligent search than keyword–based one. Although we will not bother with how to implement a search engine, we will devote this thesis to the development of an ontological semantics, including practical computational methods of extracting it from natural language texts (in limited contexts).

1.8 Survey of the chapters

In Chapter 2 we introduce the reader to the formalisms that are useful for representing either formal ontologies, natural language semantics, or both. Chapter 3

[2] Similar calculation shows that at the same time we save $100\text{W} \cdot 8 \cdot 10^5\text{h} \approx 8 \cdot 10^4\text{kWh}$ of electrical energy.

provides a general introduction to ontologies, both formal and informal ones. In Chapter 4 we explain how the notion of "types" is used in logic, ontologies, and programming languages. We provide several approaches to the merging of the formal ontologies with natural language semantics in Chapter 5. We compare our point of view with that of the state of the art in Chapter 6. The question of how to approach the same problem from the computational point of view is discussed in Chapter 7. In Chapter 8 we go one step further and try to use similar methods for dealing with unrestricted natural language. We present some final thoughts and the discussion of further work in Chapter 9. Finally, we conclude the work in Chapter 10.

Chapter 2

Logic and formalisms

Basic knowledge of various formalisms may be essential for understanding the following chapters. Therefore we will try to make the reader familiar with a set of formalisms often used for representing ontologies, conceptual feature structures, semantics of natural language sentences and phrases, etc. Such an introduction could be achieved in an alternativ fashion, i.e. by providing good references to the material that describes the formalisms needed. However, it is feared that this would be at a cost of the lack of coherence of the current text.

In addition, the standard sources describing many of these formalisms are unfortunately not rich in relevant examples. Hence, we will try to introduce the necessary background knowledge with the help of many examples.

In this chapter we will describe some of the formalisms useful for ontology engineering and semantic representation for natural language:

- First Order Predicate Logic
- Lattices
- Description Logics

We will also have a brief look at the Logic of Plurals and Mass Terms.

2.1 First Order Predicate Logic

We assume that the reader is familiar with First Order Predicate Logic, as it is a basic tool for every computer scientist. Otherwise please consult a text on the subject, e.g. [8].

Since the First Order Predicate Logic notation differs from one source to the other, we present the syntax used in the current text in Figure 2.1 on page 31.

As can be seen from the Figure, there is a syntactical distinction between predicates/functors and variables, that is variable names start with a capital letter. This is the convention adopted from the logic programming languages Prolog/Mercury. This convention eases translating a logical specification to a programming language implementation. It also helps understanding formulae in case when we use implicit quantification.

We will assume throughout the text an implicit universal quantification of variables. Consider for instance:

$$sibling(X, Y) \rightarrow sibling(Y, X)$$

In this formula two variables appear free. However, by our convention we consider them to be universally quantified, as in:

$$\forall X(\forall Y(sibling(X, Y) \rightarrow sibling(Y, X)))$$

This convention eliminates any free variables from any formula[1], in effect turning any formula into a closed one.

Please observe that the syntax is pure, in the sence that it does not allow any mathematical contaminations. In particular, the symbol of equality (=) is not inherently part of First Order Predicate Logic. Instead, one can use the following sentence for testing whether two ground terms are identical:

$$equal(X, X) \tag{2.1}$$

Notice that this definition is in fact only describing identity of two terms, not their equality. In order to introduce equality, one could e.g. use an extended version of the logic itself, which incorporates equality into the semantics of it. An example of such a logic is First Order Predicate Logic With Equality.

2.1.1 Predication

First Order Predicate Logic expresses properties of and relationships between individuals. However, we need also to express properties of relationships and relationships between relationships (e.g. that *part_of* is transitive). When reasoning about properties of various binary predicates, we shall use the following First Order Predicate Logic notation for expressing them:

$$r(\phi, \rho, \psi)$$

[1] except submormulas

$$
\begin{array}{rcl}
\langle \text{formula} \rangle & ::= & \langle \text{predicate} \rangle (\langle \text{termlist} \rangle) \\
& | & \langle \text{predicate} \rangle \\
& | & \neg \langle \text{formula} \rangle \\
& | & \langle \text{formula} \rangle \vee \langle \text{formula} \rangle \\
& | & \langle \text{formula} \rangle \wedge \langle \text{formula} \rangle \\
& | & \langle \text{formula} \rangle \rightarrow \langle \text{formula} \rangle \\
& | & \langle \text{formula} \rangle \leftarrow \langle \text{formula} \rangle \\
& | & \langle \text{formula} \rangle \leftrightarrow \langle \text{formula} \rangle \\
& | & \langle \text{quantifier} \rangle \langle \text{variablelist} \rangle (\langle \text{formula} \rangle) \\
& | & (\langle \text{formula} \rangle) \\
\\
\langle \text{quantifier} \rangle & ::= & \forall | \exists \\
\\
\langle \text{term} \rangle & ::= & \langle \text{variable} \rangle \\
& | & \langle \text{functor} \rangle \\
& | & \langle \text{functor} \rangle (\langle \text{termlist} \rangle) \\
\\
\langle \text{termlist} \rangle & ::= & \langle \text{term} \rangle \\
& | & \langle \text{term} \rangle, \langle \text{termlist} \rangle \\
\langle \text{variablelist} \rangle & ::= & \langle \text{variable} \rangle \\
& | & \langle \text{variable} \rangle, \langle \text{variablelist} \rangle \\
\\
\langle \text{lowercase} \rangle & ::= & a|b|\ldots|x|y|z \\
\langle \text{uppercase} \rangle & ::= & A|B|\ldots|X|Y|Z \\
\langle \text{letters} \rangle & ::= & \epsilon \\
& | & \langle \text{lowercase} \rangle \langle \text{letters} \rangle \\
& | & \langle \text{uppercase} \rangle \langle \text{letters} \rangle \\
\\
\langle \text{variable} \rangle & ::= & \langle \text{uppercase} \rangle \langle \text{letters} \rangle \\
\langle \text{predicate} \rangle & ::= & \langle \text{lowercase} \rangle \langle \text{letters} \rangle \\
\langle \text{functor} \rangle & ::= & \langle \text{lowercase} \rangle \langle \text{letters} \rangle
\end{array}
$$

Figure 2.1: Abstract syntax of First Order Predicate Logic used throughout the text.

Here r is a distinct predicate, expressing that ϕ is related to ψ by relation ρ. For instance, we could express that *liver* is *part_of human_body* by

$$r(liver, part\_of, human\_body)$$

We can now quantify over predicates.

When we would like to conceive of the relation as a set of pairs, we can use the following mathematical definition for arbitrary relation ρ:

$$\rho = \{(X, Y) | r(X, \rho, Y)\} \tag{2.2}$$

Please observe that this set notation is a mathematical metalanguage, not part of First Order Predicate Logic. Similarly, we can use the following mathematical definition for arbitrary relation ρ to obtain the set A_ρ of elements on which the relation is defined:

$$A_\rho = \{X | \exists Y(r(X, \rho, Y) \vee r(Y, \rho, X))\} \tag{2.3}$$

An alternative First Order Predicate Logic representation is to use a new predicate for each relation, as in:

$$part\_of(liver, human\_body)$$

This has the disadvantage that general reasoning about relations would require higher order predicates. The notation proposed resembles more natural language and mathematical notation, where most relations are used in infix form. Nevertheless, for the implementation in logic programming languages we use both notations, depending on which one is more convenient.

In the proposed notation it is easy to define general properties of relations. The following definitions are applied throughout the text:

$$\begin{aligned}
reflexive(R) &\leftrightarrow \forall X \forall Y(r(X, R, Y) \rightarrow r(X, R, X) \wedge r(Y, R, Y)) \\
antisymmetric(R) &\leftrightarrow \forall X \forall Y(r(X, R, Y) \wedge r(Y, R, X) \rightarrow r(X, equal, Y)) \\
transitive(R) &\leftrightarrow \forall X \forall Y(r(X, R, Y) \wedge r(Y, R, Z) \rightarrow r(X, R, Z))
\end{aligned}$$

We can now use these general definitions to express properties of different relations, say by declaring:

$$transitive(part\_of)^2$$

[2]This particular statement about parthood as such can be disputed. Please refer to [21] for a more detailed ontological discussion of the `part_of` relation.

Please observe that the following definition of reflexivity is deficient:

$$reflexive(R) \leftrightarrow \forall X(r(X,R,X)) \tag{2.4}$$

The problem is that we should require reflexivity to hold only for objects, which in fact engage in the relation, not for all objects in the universe.

2.1.2 Algebraic operations

Algebra plays an important role in representing the meaning of natural language. [28] We shall use the following First Order Predicate Logic notation for expressing algebraic operations:

$$o(\phi,\omega,\psi,\gamma)$$

Here o is a distinct predicate intended for expressing that ω applied to ϕ and ψ results in γ. E.g. we could express that Peirce product [3] of *has_color* relation and *green* is *green_thing* by:

$$o(has\_color,:,green,green\_thing)$$

We can use the following metalanguage definition for arbitrary operation ω to obtain the set A_ω of elements on which the operation is defined:

$$A_\omega = \{X|\exists Y \exists Z(o(X,\rho,Y,Z) \vee o(Y,\omega,X,Z))\} \tag{2.5}$$

Similarly as for binary relations, it is easy to define general properties of algebraic operations. The following is assumed to hold throughout the text:

$$\begin{aligned}
idempotent(O) &\leftrightarrow \forall(o(X,O,Y,Z) \rightarrow \\
&\quad o(X,O,X,X) \wedge o(Y,O,Y,Y)) \\
commutative(O) &\leftrightarrow \forall(o(X,O,Y,Z) \rightarrow o(Y,O,X,Z)) \\
associative(O) &\leftrightarrow \forall(o(Y,O,Z,W) \wedge o(X,O,W,V) \\
&\quad \wedge o(X,O,Y,U) \rightarrow o(U,O,Z,V)) \\
absorptive(O_1,O_2) &\leftrightarrow \forall(o(X,O_2,Y,U) \rightarrow o(X,O_1,U,X))
\end{aligned}$$

We can now use these general definitions to express properties of different operations, e.g. commutativity of set intersection:

$$commutative(\cap)$$

[3] Peirce product algebraic operation is described in section 2.3.7 on page 47.

2.1.3 Distinction between classes and instances

In many knowledge domains, e.g. in the biomedical domain we do not see a clear need for distinguishing between classes (descriptions of groups of individuals) and instances (individuals). For example, we could have a class *liver*, which represents a set of all possibly imaginable livers, and an instance $John\_Smith's\_liver$, representing a particular liver.

Since in the present framework we are concerned with understanding what sentences in biomedical papers are about, we can forget about instances, as articles rarely talk about some particulars. The general approach is to describe things more generally. Even if the article talks about a particular patient, her name will not be repeated in every sentence. Since we do not analyze the context, but analyze each sentence individually, the objects will still represent classes more than instances. Consider the following example:

This paper is about John Smith. His liver stores glucose. (2.6)

Even though the paper talks about some specific liver, our system will not be able to notice that, because it analyzes each sentence independently.

Hence in the ontological analysis described later, we do not make any use of instances, or particulars, but we only use classes. Anyway, a particular object can be conceived as a singleton set comprising that object, so an instance can be represented by a singleton class.

For further discussion of the notion of instances and classes, please refer to [47, 32].

2.2 Lattices

One of the formalisms that can be used to represent the classification within knowledge domains (and hence ontologies) is lattices. First we introduce necessary definitions, next we will present examples ilustrating the usefulness of lattices. We choose to present lattices within our chosen framework of First Order Predicate Logic.

2.2.1 Posets

We call a binary relation R a poset (partially ordered set) iff it satisfies three conditions:

$$poset(R) \leftrightarrow reflexive(R) \wedge antisymmetric(R) \wedge transitive(R)$$

Following [38], from a mathematical point of view a poset is a pair (P, R) where P is a set and R is a relation (set of pairs: $R \subseteq P \times P$) such that R is a partial order (it is reflexive, antisymmetric and transitive).

We will however use the notion of a poset and partial order simultaneously for a relation ρ, since we can always obtain such a mathematical view of a poset (A_ρ, ρ) using definitions 2.3 and 2.2 on page 32.

Let us consider the following example defining the relation *ipo* (is part of), e.g. *peru ipo southAmerika*, but *amazon ipo* both *peru* and *brasil*:

$$
\begin{gathered}
r(peru, ipo, peru) \\
r(brasil, ipo, brasil) \\
r(southAmerika, ipo, southAmerika) \\
r(amazon, ipo, southAmerika) \\
r(brasil, ipo, southAmerika) \\
r(peru, ipo, southAmerika) \\
r(amazon, ipo, brasil) \\
r(amazon, ipo, amazon) \\
r(amazon, ipo, peru)
\end{gathered}
$$

Using definition 2.3 we can obtain a mathematical view of the set A_{ipo} on which the relation is defined:

$$A_{ipo} = \{amazon, peru, brasil, southAmerika\}$$

Similarly, we can calculate the mathematical view of the relation *ipo* as a set of pairs using definition 2.2:

$$
\begin{aligned}
ipo = \{ \quad & (amazon, amazon), (peru, peru), (brasil, brasil), \\
& (southAmerika, southAmerika), \\
& (amazon, southAmerika), (brasil, southAmerika), \\
& (peru, southAmerika), (amazon, brasil), (amazon, peru) \quad \}
\end{aligned}
$$

Hence, we can refer to relation *ipo* as both a partial order and a poset.

It is easy to verify that:

$$poset(ipo)$$

2.2.2 Top and bottom

"Top" is the largest element in a given ordering, and "bottom" is the lowest one. We can define the top element T (denoted $\top$) for poset relation R as:

$$r(T, top, R) \leftrightarrow \forall X, Y(r(X, R, Y) \rightarrow r(X, R, T) \wedge r(Y, R, T))$$

Similarly, the bottom element B (denoted $\bot$) can be defined as:

$$r(B, bottom, R) \leftrightarrow \forall X, Y(r(X, R, Y) \rightarrow r(B, R, X) \wedge r(B, R, Y))$$

As an example for relation *ipo* defined in section 2.2.1, we have:

$$r(southAmerika, top, ipo) \wedge r(amazon, bottom, ipo)$$

There are poset relations R without $\top$ and $\bot$ elements:

$$\exists R(\neg\exists T(r(T, top, R)))$$
$$\exists R(\neg\exists B(r(B, bottom, R)))$$

2.2.3 Upper and lower bounds

We say that elements X and Y in poset relation R have an upper bound U, a least upper bound (supremum) S, lower bound L, and a greatest lower bound (infimum) I when:

$$\begin{aligned}
upperBound(X, Y, R, U) &\leftrightarrow r(X, R, U) \wedge r(Y, R, U) \\
supremum(X, Y, R, S) &\leftrightarrow \forall U(upperBound(X, Y, R, U) \rightarrow r(S, R, U)) \\
lowerBound(X, Y, R, L) &\leftrightarrow r(L, R, X) \wedge r(L, R, Y) \\
infimum(X, Y, R, I) &\leftrightarrow \forall L(lowerBound(X, Y, R, L) \rightarrow r(L, R, I))
\end{aligned}$$

Example:

$$upperBound(peru, amazon, ipo, southAmerika)$$
$$upperBound(peru, amazon, ipo, peru)$$
$$supremum(peru, amazon, ipo, peru)$$
$$\neg supremum(peru, amazon, ipo, southAmerika)$$
$$lowerBound(brasil, southAmerika, ipo, amazon)$$
$$lowerBound(brasil, southAmerika, ipo, brasil)$$
$$infimum(brasil, southAmerika, ipo, brasil)$$
$$\neg infimum(brasil, southAmerika, ipo, amazon)$$

2.2.4 Lattice as poset

We say that poset R is a lattice iff:

$$lattice(R) \leftrightarrow \forall X, Y(r(X, R, Y) \rightarrow \exists S(supremum(X, Y, R, S)) \wedge \exists I(infimum(X, Y, R, I)))$$

We define *boundedLattice* as:

$$boundedLattice(R) \leftrightarrow lattice(R) \wedge \exists T(r(T, top, R)) \wedge \exists B(r(B, bottom, R))$$

2.2.5 Hasse diagrams

The graphical representation of a poset known as Hasse diagram can be drawn in the following manner (from [38]): if $x \leq y$ then we place x below y and draw a line from x to y, except that lines following from reflexivity and transitivity are omitted.

The Hasse diagram of the poset *ipo* (which happens to be a lattice) is presented in figure 2.2 on page 38.

2.2.6 *isa* as partial order

Of particular interest is the *isa* relation, which forms a poset when coupled with arbitrary set of classes. We can hence easily represent the taxonomy of our domain using the Hasse diagram.

$$poset(isa)$$

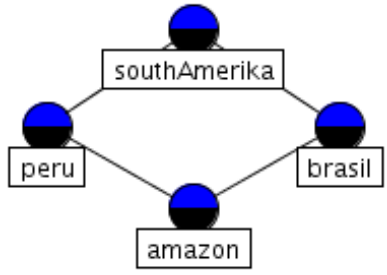


Figure 2.2: Hasse diagram for *ipo* lattice.

We might insist that the *isa* relation is a lattice in order to enjoy the stated mathematical properties for our classification.

$$lattice(isa)$$

We present a sample Hasse diagram for the *isa* relation in our domain in figure 2.3 on page 39.

2.2.7 Lattices as algebras

As defined in [38], from mathematical point of view, lattice is a triple (L, J, M), where L is a nonempty set, J and M are binary operations defined on L, having special properties. In our First Order Predicate Logic notation, however, the set L is implicitly defined and can be obtained using definition 2.5.

We shall call a pair of operations J, M a lattice, when:

$$\begin{aligned} lattice(J, M) \leftrightarrow & idempotent(J) \wedge idempotent(M) \wedge \\ & commutative(J) \wedge commutative(M) \wedge \\ & associative(J) \wedge associative(M) \wedge \\ & absorptive(J, M) \wedge absorptive(M, J) \end{aligned}$$

We usually refer to the two operations as join (denoted $\vee$) and meet (denoted $\wedge$). Symbols $\vee$ and $\wedge$ are reserved for disjunction and conjunction in First Order Predicate Logic, hence we shall avoid them in our formulae.

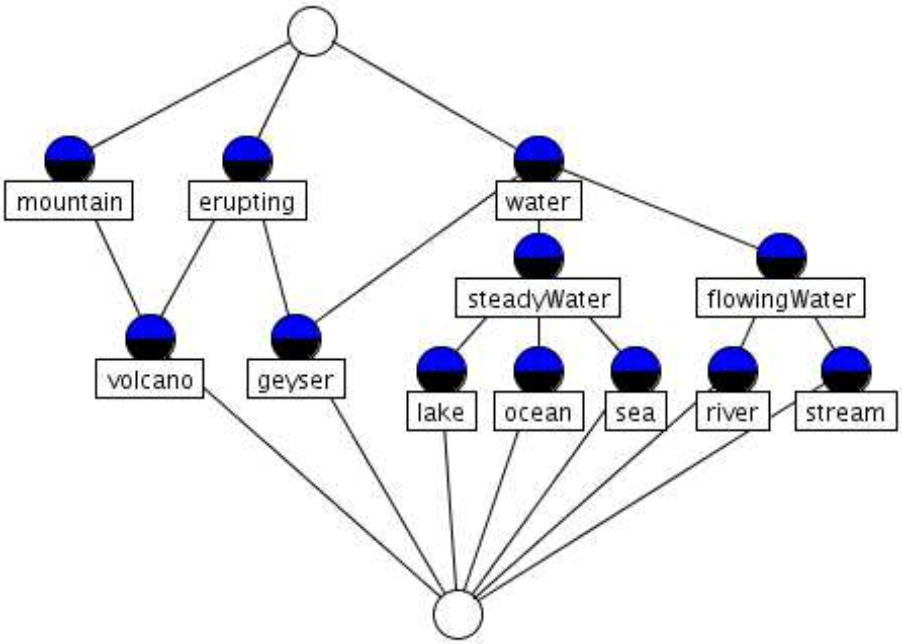


Figure 2.3: Hasse diagram for *isa* lattice.

2.2.8 Dual nature of lattices

The two definitions below allow us to transform a lattice defined as partial order (section 2.2.4) into lattice defined as an algebra (section 2.2.7) and vice versa. The proof can be found in [38].

$$\begin{aligned} lattice(R) &\rightarrow \forall(supremum(X,Y,R,S) \rightarrow o(X,join,Y,S)) \\ &\quad \wedge\forall(infimum(X,Y,R,I) \rightarrow o(X,meet,Y,I)) \\ &\quad \wedge lattice(join,meet) \\ lattice(J,M) &\rightarrow \forall(o(X,J,Y,X) \rightarrow r(X,\leq,Y)) \wedge lattice(\leq) \end{aligned}$$

2.2.9 Atoms

A node A is a lattice atom iff:

$$\begin{aligned} atom(A,R) \leftrightarrow & lattice(R) \wedge \\ & \forall(r(X,R,A) \rightarrow equal(X,A) \vee (equal(X,B) \wedge r(B,bottom,R))) \end{aligned}$$

2.2.10 Algebraic lattices and classifications

Thanks to the algebraic nature of lattices (section 2.2.7), we can use meet and join algebraic operations to specify our domain's classification as a lattice.

As an example, consider the *ipo* relation, depicted in figure 2.2 on page 38. We could specify this lattice using the following:

$$\begin{aligned} & o(brasil,join,peru,southAmerica) \\ & o(brasil,meet,peru,amazon) \\ & lattice(join,meet) \end{aligned}$$

We do not need to specify anything else, e.g. $o(amazon,meet,peru,amazon)$, as it follows from lattice properties (defined in section 2.2.7).

2.3 Description Logics

2.3.1 Restricting First Order Predicate Logic

As described in [5] First Order Predicate Logic has some drawbacks when it comes to applying it in knowledge representation systems: many problems are

undecidable and most of the decidable ones are intractable. Description Logics might serve as a useful replacement. Description Logics is very much restricted compared to First Order Predicate Logic as far as the expressive power is concerned. This means that sometimes we may have difficulties expressing something we want in Description Logics. It is on the other hand decidable and computationally efficient.

The so–called Peirce product (described further in [12] and in Section 2.3.7 on page 47) is the link between lattices and Description Logics.

2.3.2 TBox

The terminology of the domain in question is described in the so–called TBox. The terminology consists of concepts and roles. Concepts are equivalent to unary predicates in First Order Predicate Logic (they can represent classes), while roles are equivalent to binary predicates (they represent relations). Atomic concepts and roles are common to all Description Logics languages, while they differ by how expressive is the construction of complex ones.

Typical systems allow to perform various forms of reasoning based on the terminology introduced by the TBox, e.g. whether the specification is satisfiable or whether one description subsumes the other.

Most Description Logics languages are subsets of the $\mathcal{ALCN}$ language. They provide a set of constructors, which allows to describe complex concepts in terms of atomic ones.

The following BNF rules describe the available $\mathcal{ALCN}$ constructors and their syntax (where ⟨atomic concept⟩ is an arbitrary atomic concept identifier, ⟨atomic role⟩ is an arbitrary atomic role, $\top$ is the universal concept, $\bot$ is the

bottom concept and ⟨n⟩ is a natural number):

$$
\begin{array}{rcl}
\langle \text{concept} \rangle & ::= & \langle \text{atomic concept} \rangle \\
& | & \top \\
& | & \bot \\
& | & \neg \langle \text{concept} \rangle \\
& | & \langle \text{concept} \rangle \sqcap \langle \text{concept} \rangle \\
& | & \langle \text{concept} \rangle \sqcup \langle \text{concept} \rangle \\
& | & \forall \langle \text{atomic role} \rangle . \langle \text{concept} \rangle \\
& | & \exists \langle \text{atomic role} \rangle . \langle \text{concept} \rangle \\
& | & \geqslant \langle \text{n} \rangle \langle \text{atomic role} \rangle \\
& | & \leqslant \langle \text{n} \rangle \langle \text{atomic role} \rangle \\
\\
\langle \text{terminological axiom} \rangle & ::= & \langle \text{concept} \rangle \sqsubseteq \langle \text{concept} \rangle \\
& | & \langle \text{concept} \rangle \equiv \langle \text{concept} \rangle \\
\langle \text{definition} \rangle & ::= & \langle \text{atomic concept} \rangle \sqsubseteq \langle \text{concept} \rangle \\
& | & \langle \text{atomic concept} \rangle \equiv \langle \text{concept} \rangle
\end{array}
$$

A set of ⟨definition⟩s is called a TBox if each symbolic name is defined at most once. This limitation however seems to be abandoned in recent versions of some Description Logics reasoning implementations.

2.3.3 ABox

The ABox provides set of assertions expressed in terms of terminology defined by the TBox. There are only two types of assertions one can make: concept assertions of the form $C(a)$ and role assertions of the form $R(a, b)$, where C is a concept, R is a role, and a, b are names of individuals.

Existing systems allow checking whether assertions expressed by the ABox are consistent and whether a particular individual is an instance of a given concept.

2.3.4 Translating from Description Logics to First Order Predicate Logic

Let us define a function $[\cdot]$, which given $\mathcal{ALCN}$ formula, computes a First Order Predicate Logic formula with equivalent meaning. This is usually done resorting to lambda calculus, while we present an novel method using First Order Predicate Logic only:

$$
\begin{aligned}
[A] &= A(X) \text{ for } A \text{ being atomic concept} \\
[\top] &= true \\
[\bot] &= false \\
[\neg C] &= \neg [C] \\
[C_1 \sqcap C_2] &= [C_1] \wedge [C_2] \\
[C_1 \sqcup C_2] &= [C_1] \vee [C_2] \\
[\forall R.C] &= \forall Y (R(X,Y) \rightarrow \forall X (equal(Y,X) \rightarrow [C])) \\
[\exists R.C] &= \exists Y (R(X,Y) \wedge \forall X (equal(Y,X) \rightarrow [C])) \\
[\geqslant n\ R] &= \exists Y_1 \cdots \exists Y_n \left(R(X,Y_1) \wedge \ldots \wedge R(X,Y_n) \wedge \bigwedge_{i<j} \neg equal(Y_i,Y_j) \right) \\
[\leqslant n\ R] &= \forall Y_1 \cdots \forall Y_{n+1} \left(R(X,Y_1) \wedge \ldots \wedge R(X,Y_{n+1}) \rightarrow \bigvee_{i<j} equal(Y_i,Y_j) \right) \\
[C_1 \sqsubseteq C_2] &= [C_1] \rightarrow [C_2] \\
[C_1 \equiv C_2] &= [C_1] \leftrightarrow [C_2]
\end{aligned}
$$

The definitions are not all trivial. Let us consider why we need $\forall X (equal(Y,X) \rightarrow [C])$. $[C]$ is a First Order Predicate Logic formula, having at most one free variable, X. If we quantify it universally and add another free variable Y, and require that $[C]$ holds provided $equal(Y,X)$, then we obtain a simple renaming of variables (from X to Y). Therefore $\forall X (equal(Y,X) \rightarrow [C])$ is like $[C]$, except that the free variable has been renamed from X to Y.

As we can transform each $\mathcal{ALCN}$ formula into First Order Predicate Logic formula, the notions of interpretation, model, satisfiability, etc. carry over from First Order Predicate Logic to Description Logics.

For more information about semantics of Description Logics, please consult [5].

2.3.5 Examples

Below we present a TBox defining a taxonomy analogous to the Hasse diagram presented earlier in figure 2.3 on page 39. We assume that $mountain$, $water$, $flowingWater$, $erupting$ are atomic base concepts and $hasTributary$ is an atomic role.

$$
\begin{aligned}
volcano &\equiv mountain \sqcap erupting \\
geyser &\equiv water \sqcap erupting \\
steadyWater &\equiv water \sqcap \neg flowingWater \sqcap \neg geyser \\
lake &\sqsubseteq steadyWater \\
ocean &\sqsubseteq steadyWater \\
sea &\sqsubseteq steadyWater \\
river &\equiv water \sqcap flowingWater \sqcap \exists hasTributary.\top \\
stream &\equiv water \sqcap flowingWater \sqcap \forall hasTributary.\bot \\
largeRiver &\equiv river \sqcap \geqslant 2\ hasTributary
\end{aligned}
$$

Let us use the function f defined in section 2.3.4 in order to compute an

equivalent First Order Predicate Logic formulae. We get:

$$
\begin{array}{rcl}
volcano(X) & \leftrightarrow & mountain(X) \wedge erupting(X) \\
geyser(X) & \leftrightarrow & water(X) \wedge erupting(X) \\
steadyWater(X) & \leftrightarrow & water(X) \wedge \neg flowingWater(X) \wedge \neg geyser(X) \\
lake(X) & \rightarrow & steadyWater(X) \\
ocean(X) & \rightarrow & steadyWater(X) \\
sea(X) & \rightarrow & steadyWater(X) \\
river(X) & \leftrightarrow & water(X) \wedge flowingWater(X) \\
 & & \wedge \exists Y(hasTributary(X,Y) \wedge \forall X(equal(Y,X) \rightarrow true)) \\
stream(X) & \leftrightarrow & water(X) \wedge flowingWater(X) \\
 & & \wedge \forall Y(hasTributary(X,Y) \rightarrow \forall X(equal(Y,X) \rightarrow false)) \\
largeRiver(X) & \leftrightarrow & river(X) \wedge \\
 & & \exists Y_1 \exists Y_2 \exists Y_3(hasTributary(X,Y_1) \wedge \\
 & & \qquad hasTributary(X,Y_2) \wedge \\
 & & \qquad hasTributary(X,Y_3) \wedge \\
 & & \qquad \neg equal(Y_1,Y_2) \wedge \neg equal(Y_1,Y_3) \wedge \neg equal(Y_2,Y_3))
\end{array}
$$

Observe the coherence with lattices, where we would have:

$$r(lake, isa, steadyWater)$$

We can rewrite the formulae defining *river* and *stream* into equivalent shorter versions:

$$
\begin{array}{rcl}
river(X) & \leftrightarrow & water(X) \wedge flowingWater(X) \wedge \exists Y(hasTributary(X,Y)) \\
stream(X) & \leftrightarrow & water(X) \wedge flowingWater(X) \wedge \neg \exists Y(hasTributary(X,Y))
\end{array}
$$

Below we present a sample ABox for geographical domain:

$$
\begin{aligned}
&mountain(chomolungma)\\
&volcano(vesuvius)\\
&geyser(strokkur)\\
&lake(caspianSea)\\
&sea(baltic)\\
&ocean(pacific)\\
&stream(branco)\\
&stream(jurua)\\
&stream(nanay)\\
&river(amazon)\\
&hasTributary(amazon, branco)\\
&hasTributary(amazon, jurua)\\
&hasTributary(amazon, nanay)\\
&largeRiver(amazon)
\end{aligned}
$$

In reality *branco*, *jurua* and *nanay* are *river*s, but we pretend they have no tributaries, hence they must be *stream*s in order to avoid inconsistency with our TBox.

2.3.6 Reasoning

Description Logics is very restricted when compared to First Order Predicate Logic. However, this allows Description Logics to be not only decidable, but also tractable, or even very efficiently decidable for some of its variants. Many of–the–shelf reasoners exist. They provide many useful forms of reasoning about TBoxes and ABoxes, e.g. they can verify the consistency of a logical theory, or perform a model checking, where an ABox would be verify against a given TBox.

The logical specification in itself would not be that interesting if it would not allow us to automatically draw some conclusions. For instance, using our specification for the geographical domain, we can conclude that:

$$river \sqcap stream \equiv \bot$$

2.3.7 Peirce Algebra

Peirce algebra is a two–sorted algebra, which combines operations on sets and relations. Hence, it is very useful for ontological applications, where classes can be conceived as sets, and where relations play major role. A very detailed introduction is available in [12].

Of particular interest for us is the Peirce product, as the conceptual feature structures used for representing parts of the ontological semantics are based on the it. Peirce product (:) is defined as:

$$\rho : \phi = \{X | \exists Y((X, Y) \in \rho \ \wedge \ Y \in \phi)\},$$

where ρ is a relation and ϕ is a class understood as a set. Notice that the Peirce product has an equivalent representation in Description Logics:

$$\exists \rho.\phi$$

Hence, the Peirce product can be considered to be the link between lattices and Description Logics.

2.4 Logic of Plurals and Mass Terms

The Logic of Plurals and Mass Terms is a variant of First Order Predicate Logic. It defines special facilities for describing plural formations. The most interesting of those is the collective operator (denoted $\oplus$), written by convention in an infix notation. This operator takes two arguments and creates a collective object, representing the plural entity consisting of both arguments.

Let us have a look at some sentences, so that we can explain how $\oplus$ works by example.

2.4.1 "A boy and a girl played."

This sentence has two main readings:

- The collective reading, where the boy and the girl were engaged in a common play.
- The distributive reading, where they both played, but individually.

Logic of Plurals and Mass Terms

In Logic of Plurals and Mass Terms we can utilize the collective operator to represent the collective reading:

$$\exists X \exists Y (boy'(X) \wedge girl'(Y) \wedge played'(X \oplus Y))$$

On the other hand, no special facilities are called for if we want to represent the distributive reading:

$$\exists X \exists Y (boy'(X) \wedge girl'(Y) \wedge played'(X) \wedge played'(Y))$$

Davidsonian view in First Order Predicate Logic

In the davidsonian view, we make use of an additional variable for representing the event of playing. So the action of playing will not be represented by the predicate $played'(\ldots)$ anymore. This has many advantages. We do not need to use the collective operator to represent the fact that more than one agent took part in the action of playing, so we can stay within First Order Predicate Logic. Additionally, the arity of the $played'$ no longer needs to be decided upon. E.g. if we also wanted to express the time at which the action took place, we would need to extend the unary predicate $played'(who)$ to a binary one $play'(who, when)$. Such a problem is nonexistent with the davidsonian representation of the meaning of the sentence.

The collective reading takes the form:

$$\begin{aligned}\exists E \exists X \exists Y \exists T (&playing(E) \wedge boy(X) \wedge girl(Y) \\ &\wedge agt(E, X) \wedge agt(E, Y) \wedge past(T) \wedge tmp(E, T))\end{aligned}$$

The distributive reading will look as follows:

$$\begin{aligned}\exists E_1 \exists E_2 \exists X \exists Y \exists T_1 \exists T_2 (&playing(E_1) \wedge boy(X) \wedge agt(E_1, X) \wedge past(T_1) \\ &\wedge playing(E_2) \wedge girl(Y) \wedge agt(E_2, Y) \wedge past(T_2)) \\ &\wedge tmp(E_1, T_1) \wedge tmp(E_2, T_2)\end{aligned}$$

Description Logics

Finally, let us have a look at how such a semantics can be represented in Description Logics.

Here we will follow our previous representation using the davidsonian view. We chose our semantics to represent a concept rather than a terminological axiom. The concept will be the one corresponding to the event that happened.

For the collective reading, we get the following single concept:

$$playing \sqcap \exists agt.boy \sqcap \exists agt.girl \sqcap \exists tmp.past$$

And for the distributive reading we get two separate semantics concepts (as we have two different events):

$$playing \sqcap \exists agt.boy \sqcap \exists tmp.past$$

and

$$playing \sqcap \exists agt.girl \sqcap \exists tmp.past$$

2.4.2 "George and Martha met."

This sentence has only the collective meaning. It is very hard to imagine that somebody would use such a sentence with the intention of saying that both George and Martha met somebody, but not each other.

Logic of Plurals and Mass Terms

The reading becomes (with the help of the collective operator):

$$met'(george \oplus martha)$$

Davidsonian view, First Order Predicate Logic

We can again utilize the davidsonian view for the purpose of representing the collective reading within First Order Predicate Logic:

$$\begin{aligned} \exists E \exists X \exists Y \exists T (&meeting(E) \wedge george(X) \wedge martha(Y) \\ &\wedge agt(E, X) \wedge agt(E, Y) \wedge past(T) \wedge tmp(E, T)) \end{aligned}$$

Description Logics

Following our approach from the previous sentence, the reading becomes:

$$meeting \sqcap \exists agt.george \sqcap \exists agt.martha \sqcap \exists tmp.past$$

Chapter 3

Introduction to ontologies

This chapter provides a short general introduction to ontologies, both formal and informal ones. It is not the intention of the author that the chapter is a rewrite of some books, but rather a brief introduction, which refers to certain sources. The aim of this chapter is specifically to establish how ontological notions (such as classes, properties, relations, etc.) can be represented easily in a formal way.

In focus is the representation of ontologies in First Order Predicate Logic. This is due to the fact that if one can express ontological data in First Order Predicate Logic, it is less difficult to adjust that representation to fit Prolog at the implementation stage. Various First Order Predicate Logic representations can be compared, so that it becomes evident how one can later implement the needed reasoning machinery easily and efficiently.

This chapter also discusses various means of simplifying the view of the world, e.g. lifting instances to singleton classes, etc.

A nice introduction and overview of ontologies and their various definitions is provided in [23]. The word "Ontology" appeared thousands of years ago, where ancient philosophers tried to explain the essence of being. In such context, we use the capitalized word "Ontology". Please consult [23] for an overview of the fascinating problems that philosophers have been dealing with.

However, the current text focuses on formal ontologies, or simply ontologies, written with lower–case "o". Formal ontology can be understood as specification of a conceptualization. It is crucial that the ontology is specified in a formal way, so that it is machine–readable and understandable. A conceptualization can be thought of as a view of the world shared by some people. Of course none

of the ontologies can describe a full conceptualization of the whole world. In most cases an ontology attempts to describe only a specific domain.

3.1 Classes and instances

3.1.1 Distinction between class and instance

In most domains we clearly see the need for distinguishing between classes (descriptions of groups of individuals) and instances (individuals). For example, we could have a class *mountain*, which represents a set of all possibly imaginable mountains, and an instance *chomolungma*, representing a particular mountain.

In concrete domains (as opposed to abstract ones), we insist that every instance represents an individual, which exists physically. On the other hand the class *mountain* contains also non–existent objects. E.g. it comprises mountains having 9000m of height, even though no such mountains exist on Earth.[1] If such a mountain existed in history or will appear in the future, we may construct an instance representing it without modifying the class. Modifying the class means changing the relations in which the class engages, e.g. *isa* relation by adding or removing a pair of the form $r(\phi, isa, \psi)$ from the knowledge base, for arbitrary ϕ and ψ. Such a way of defining classes by their properties is known as intensional, as opposed to extensional manner, where classes are defined simply by listing all elements.

Hence, once classes are defined and relations between them established, they represent many different worlds. We can, by introducing specific instances, represent a particular world, e.g. planet Earth. We could represent a different world, e.g. planet Mars by using the same classes, yet different instances.

This is strongly related to the distinction between the TBox and the ABox in Description Logics.

3.1.2 Inheritance and *isa*

Let us consider inheritance of properties from classes to subclasses. From intensional perspective, all elements in a subclass need to possess all the properties possessed jointly by all the members of parent class. Inheritance expresses the fact that some classes are generalizations of other classes. The more general class is referred to as superclass or base class, and the less general one as

[1] Perhaps it also comprises the Meinong's golden mountain – a common literature example for such non–existent objects. [14]

subclass or derived class. If the superclass represents a set of objects O, the subclass represents a subset of O, which is in accord with the extensional view of inheritance.

Inheritance is represented by the relation *isa* in the following manner:

$$r(flowingWaterObject, isa, waterObject)$$

We define *isa* to be a transitive relation:

$$transitive(isa)$$

Hence from the knowledge base, say:

$$r(flowingWaterObject, isa, waterObject)$$
$$r(river, isa, flowingWaterObject)$$

we can deduce:

$$r(river, isa, waterObject)$$

Hence, all rivers will need to possess the properties of water objects, e.g. that they are made of water, etc.

3.1.3 Relation *instanceof*

The relation between classes and instances is captured by the *instanceof* relation. We have:

$$r(chomolungma, instanceof, mountain)^2$$

It is important to distinguish between *isa* and *instanceof* relations. In the natural language these are often confused, as in the following example:

$$r(river, isa, flowingWaterObject)$$
$$r(orinoco, isa, river)$$

Such a usage is flawed, as *river* represents a class comprising all rivers and it can be instantiated. On the contrary *orinoco* is not a class, as it represents an individual, and it cannot be instantiated.

The examples in Section 3.1.4 also ilustrate that *instanceof* is not transitive (while *isa* is).

Furthermore, the following rule (defined as *instance inference* in [47]) applies:

$$\forall(r(I, instanceof, C_1) \wedge r(C_1, isa, C_2) \rightarrow r(I, instanceof, C_2))$$

[2]Notice that we use lower–cased constants for geographic/politic names due to our convention in which upper–cased names are reserved for variables.

3.1.4 Spanning objects

In [47] we learn about the need for introducing spanning objects. A spanning object is both a class and an instance of another class. In our sample geography domain, the need for those is not apparent, because the domain is not abstract. We can add abstract concepts to the domain in order to discuss spanning objects. Let us consider the class $typeOfGeographicObject$. Instances of this class could be $theRiver$, $theMountain$, etc. On the other hand we have class $mountain$ and sample instance $chomolungma$. The simplest representation for that situation would be:

$$r(theMountain, instanceof, typeOfGeographicObject)$$
$$r(chomolungma, instanceof, mountain)$$

This representation, according to [47], is flawed, because the same concept (here mountain) needs an artificial split into an instance $theMountain$ and a class $mountain$.

We could express the correlation between $theMountain$ and $mountain$ in terms of relation $abstracts$:

$$r(theMountain, abstracts, mountain)$$

In [47] we learn about problems arising from using relation $abstracts$. Individuals $theMountain$ and $chomolungma$ are fundamentaly different, because $chomolungma$ is a physical object in Geography domain, while $theMountain$ is not. They do not exist in the same universe of discourse and this fact should be explicitly modelled in our knowledge base.

Hence, the representation which makes use of spanning objects looks as follows:

$$r(typeOfGeographicObject, classIn, metaGeographics)$$
$$r(mountain, instanceIn, metaGeographics)$$
$$r(mountain, classIn, geographics)$$
$$r(chomolungma, instanceIn, geographics)$$

$$r(mountain, instanceof, typeOfGeographicObject)$$
$$r(chomolungma, instanceof, mountain)$$

We have to introduce two universes of discourse: $metaGeographics$ and $geographics$. $mountain$ is related to both of them, but using different relations $instanceIn$ and $classIn$, respectively.

In order to enforce correct usage of those relations in the knowledge base one could add the following conditions:

$$\forall C, U_1, U_2(r(C, classIn, U_1) \wedge r(C, classIn, U_2) \rightarrow equal(U_1, U_2))$$
$$\forall I, U_1, U_2(r(I, instanceIn, U_1) \wedge r(I, instanceIn, U_2) \rightarrow equal(U_1, U_2))$$
$$\forall S, U_1, U_2(r(S, instanceIn, U_1) \wedge r(S, classIn, U_2) \rightarrow \neg equal(U_1, U_2))$$
$$\forall I, C(r(I, instanceof, C) \rightarrow \exists U(r(I, instanceIn, U) \wedge r(C, classIn, U)))$$

3.2 Part-whole relation

Part-whole relations play an important role in scientific modeling. They express that some things are parts of other things. The graph of all part relations for a given domain is refered to as *partonomy*.

Following [20], we define the supplementation property *suppl* of disjoint elements for a relation R:

$$suppl(R) \leftrightarrow \forall A, B(r(A, R, B) \rightarrow \exists C(r(C, R, B) \wedge \neg\exists X(r(X, R, A) \wedge r(X, R, C))))$$

Then we can define $partof$ relation as a strict partial ordering relation, thus having the following properties:

$$antisymmetric(partof)$$
$$transitive(partof)$$
$$suppl(partof)$$

Based on these properties a sample relation $partof$ knowledge base could look as follows[3]:

$$\begin{aligned} r(zealand, &\ partof, &\ denmark) \\ r(jutland, &\ partof, &\ denmark) \\ r(denmark, &\ partof, &\ europe) \\ r(poland, &\ partof, &\ europe) \end{aligned}$$

[3]We use lower–cased constants for geographic/politic names due to our convention in which upper–cased names are reserved for variables.

3.3 Types of formal ontologies

Formal ontologies can be generally divided into lightweight and heavyweight ontologies. The former are expressed using a very simple formalism, usually variable–free one (like Description Logics and unlike First Order Predicate Logic), so that they serve as an overview of the conceptualization. The latter call for more complicated formalism, as they usually model the conceptualization in a very detailed manner. In heavyweight ontologies a large amount of the information is implicit and is left to be inferred by the engine, hence the name.

Another categorization of ontologies focuses on the level of reality, which they describe. This division is presented in the following sections.

3.3.1 Top–level ontologies

Top–level ontologies, as the name suggests, are supposed to describe the most general view of the world. For a nice overview of different upper level ontologies, please refer to [23].

Out of many available top–level ontologies, we have decided to focus on Basic Formal Ontology (BFO). Like most top–level ontologies, BFO divides the world into enduring entities, called continuants (e.g. the body of a person), and 4–dimensional entities, called occurrents (e.g. the life of a person). A particularly good description of BFO can be found in [43]:

> *BFO grows out of a philosophical orientation which overlaps with that of DOLCE and SUMO. Unlike these, however, it is narrowly focused on the task of providing a genuine upper ontology which can be used in support of domain ontologies developed for scientific research, as for example in biomedicine within the framework of the OBO Foundry.*

3.3.2 Domain ontologies

Domain ontologies consist of classes characteristic for a given domain. Since the current text is concerned mainly with biomedical domain texts, we shall be interested in biomedical ontologies.

There exists a large collection of biomedical ontologies, which are freely available, called Open Biomedical Ontologies Foundry (OBO) [45]. It consists of many domain–specific ontologies, gathered together in a unified format.

It is worth noticing that a domain ontology must be constructed by domain experts, not by software engineers who want to use the ontology. Hence, in the

current project such an ontology is considered to be the input to the system, rather than a part of it. Various papers focus on the topic of creating a high–quality ontologies, e.g. [42].

3.3.3 Merging top–level and domain ontologies

Merging a top–level ontology with a particular domain ontology is not always easy. Top–level ontologies are usually an attempt at a "perfect" categorization of the world around us. They might be very abstract, in contrast to the domain ontologies. The latter usually attempt to be as complete as possible, trying to include all the concepts found in a particular domain. Therefore they tend to lose the perfectionistic view of the world. It might therefore be difficult to merge the two into one, coherent ontology.

The reason why we would like to merge the two is that both are insufficient if used alone. Top–level ontologies are simply very small, normally having less than a hundred concepts. Therefore they cannot be used for analyzing natural language, where much more concepts are found. The domain ontologies are, on the other hand, relatively big, reaching even hundreds of thousands of concepts. Unfortunately, they are always focused on the domain, so they are missing the inter–domain concepts. Consider the sentence:

Insulin forces storage of glycogen. (3.1)

We can find three domain–specific concepts in the sentence: the chemical substances insulin and glycogen, and the biological process of storing a substance. Unfortunately, the concept of action of forcing something cannot be present in the biomedical ontology, as it is more of an inter–domain concept. If we can construct an ontology, which contains all of the above concepts, the ontological analysis of the sentence will be more complete.

A sample ontology, created by merging the top–level Basic Formal Ontology with common biomedical concepts is presented in Figure 3.1 on page 58.

3.4 Querying Ontologies with Prolog

Let us try to formalize an ontology with the goal of querying it in mind. We will use First Order Predicate Logic as our formalism due to its closeness to Prolog.

Whenever a greek letter ϕ or ψ is encountered in a formula, it denotes a parameter and the formula itself is a parameterized formula, which can be

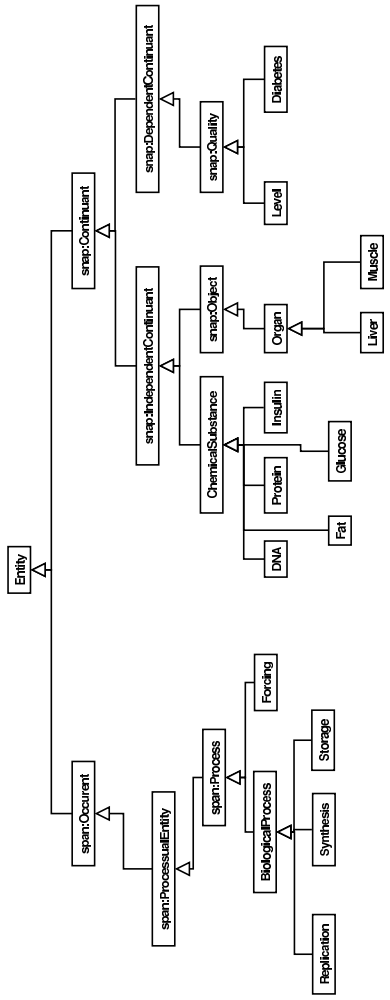


Figure 3.1: A sample ontology, created by merging the top–level Basic Formal Ontology with common biomedical concepts.

transformed into proper First Order Predicate Logic formula by instantiating the parameter with a constant or variable.

3.4.1 Pancreas Diagram

The information, which is to be modelled in our knowledge base is presented in Figure 3.2.

3.4.2 Binary relations

We will express that β_cell *secrete*s *insulin* by

$$r(\beta\_cell, secrete, insulin)$$

.

An alternative First Order Predicate Logic representation would be to use a new predicate for each relation, as in $secrete(\beta\_cell, insulin)$. This has the disadvantage that general reasoning about relations would require higher order logic. In the notation we propose it is easy to define general properties of relations.

We can now use these general definitions to express properties of different relations, e.g.:

$$transitive(isa)$$

3.4.3 *isa* defined on top of *inst*.

In [41] relation *isa* is defined on top of relation *inst* in the following way:

$$\forall(r(A, isa, B) \leftarrow \forall X(r(X, inst, A) \rightarrow r(X, inst, B)))$$

Hence, if we define *inst* in our knowledge base, we could in principle get correctly inferred *isa*. Unfortunately, prolog does not provide support for full First Order Predicate Logic reasoning, but only for definite clauses with certain extensions. Therefore in order to use this definition in Prolog, we need to rewrite it slightly:

$$\begin{aligned} & \forall A(\forall B(r(A, isa, B) \leftarrow \forall X(r(X, inst, A) \rightarrow r(X, inst, B)))) \\ \equiv \quad & \forall A(\forall B(r(A, isa, B) \leftarrow \neg\neg\forall X(\neg r(X, inst, A) \vee r(X, inst, B)))) \\ \equiv \quad & \forall A(\forall B(r(A, isa, B) \leftarrow \neg\exists X(\neg r(X, inst, A) \vee r(X, inst, B)))) \end{aligned}$$

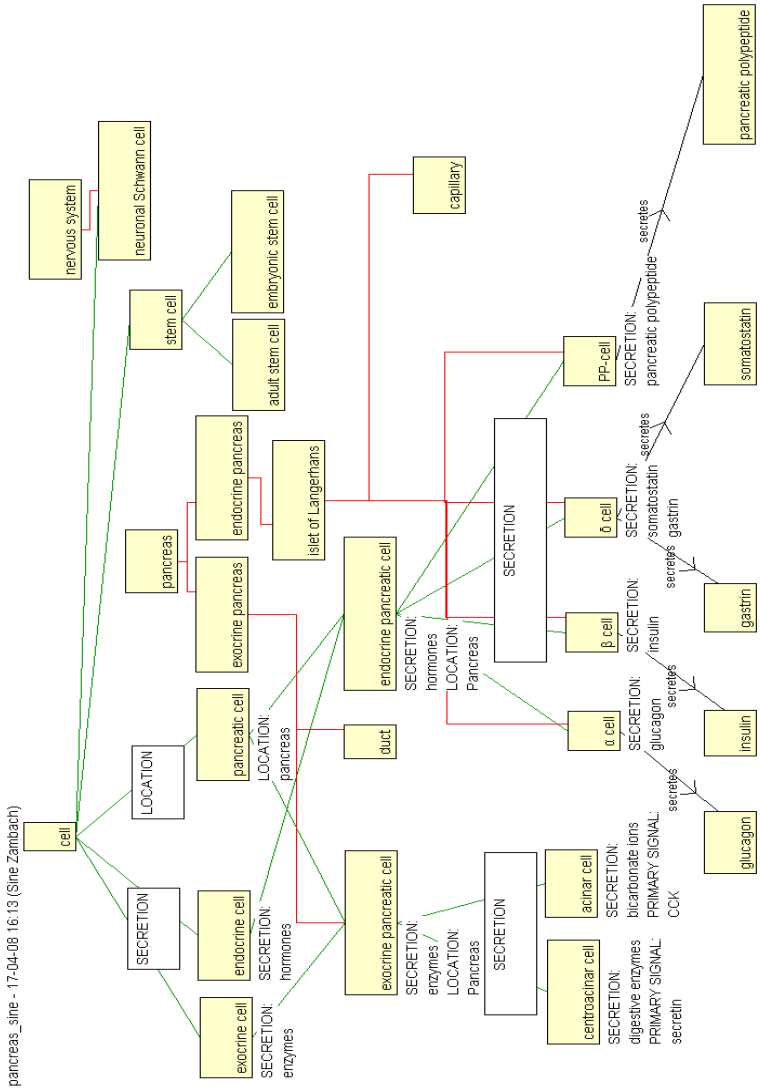


Figure 3.2: Pancreas diagram, created by Sine Zambach, Roskilde University, Denmark.

This can be redefined as two formulas:

$$\forall A(\forall B(r(A, isa, B) \leftarrow \neg isa\_counterexample(A, B)))$$
$$\forall A(\forall B(isa\_counterexample(A, B) \leftarrow \exists X(\neg r(X, inst, A) \vee r(X, inst, B))))$$

We can extract the existential quantifier from the latter formula to finally obtain:

$$\forall A(\forall B(r(A, isa, B) \leftarrow \neg isa\_counterexample(A, B)))$$
$$\forall A(\forall B(\forall X(isa\_counterexample(A, B) \leftarrow \neg r(X, inst, A) \vee r(X, inst, B))))$$

The form at which we have arrived is suitable for making inferences in Prolog, since we have two definite[4] clauses, and we can represent negation using non–provability, or negation as failure:

```
1 r(A,isa,B) :- \+ isa_counterexample(A,B).
  isa_counterexample(A,B) :- \+ r(X,inst,A), r(X,
      inst,B).
```

It is worth mentioning that in our pancreas example we have no information about any instances. It would be infeasible to provide entries of the form $r(\phi, inst, \psi)$ for all instances of the classes shown in the pancreas diagram. Hence, we need an alternative approach, which is presented in Section 3.4.4.

3.4.4 Knowledge base design

It was decided that the diagram should be translated systematically into a corresponding First Order Predicate Logic specification. The diagram has been used as the database–like input only, in the sense that a database of factual clauses has been retrieved from it using the following rules:

- Each yellow rectangle named ϕ has been translated to the following fact:

  $$class(\phi)$$

  Thus expressing existence of class ϕ in our domain of discourse.

- Each green straight line connecting a rectangle named ϕ with a rectangle named ψ has been translated to the following factual clause:

  $$r(\phi, isa, \psi)$$

[4]The clauses are not really definite, as we use negation in the body.

Those straight green lines do not have arrows indicating the direction, so the convention assumed was that ϕ is below ψ in the picture, or in other words the parent class is higher up in the taxonomy.

- Each red zigzag line connecting a rectangle named ϕ with a rectangle named ψ has been translated to the following factual clause:

$$r(\phi, partof, \psi)$$

Those red zigzag lines do not have arrows indicating the direction, so the convention assumed was that ϕ is below ψ in the picture, or in other words ψ class is higher up in the partonomy than ϕ.

In this way all the explicit knowledge visible on the pancreas diagram has been transformed to First Order Predicate Logic and Prolog. Having no biological knowledge whatsoever, we have not modified the diagram in any way.

3.4.5 Well–formedness verification

In [41] we can find the following axioms governing relations in our knowledge base:

- *inst* relation must hold between an instance and a class. We do not use *inst* in our system, but we use relations between classes that are defined on top of it in [41]. Hence, we can add the following axiom:

$$\begin{aligned} error() \leftarrow \exists R(\exists X(\exists Y(& \quad (r(X,R,Y) \wedge \neg db\_relation(R)) \\ & \vee \quad (r(X,R,Y) \wedge \neg db\_class(X)) \\ & \vee \quad (r(X,R,Y) \wedge \neg db\_class(Y))))) \end{aligned}$$

This can be rewritten in Prolog as:

```
error(undefined_relation(R)) :- r(_,R,_),
    \+ db_relation(R).
error(undefined_class(X)) :- r(X,_,_), \+
    db_class(X).
error(undefined_class(X)) :- r(_,_,X), \+
    db_class(X).
```

Now, we can run the query asking for an error:

```
?- error(E).
E = undefined_class(enzyme) ;
E = undefined_class(hormone) ;
E = undefined_class(digestive_enzyme) ;
E = undefined_class(secretin) ;
E = undefined_class(bicarbonate_ion) ;
E = undefined_class(cck) ;
No
```

We can see that some things have been used on the pancreas diagram without being put in the yellow box. In order to make the knowledge base consistent, we have added them as classes. One would argue that we should also define various relations after adding those classes, e.g. that $db\_r(somatostatin, isa, hormone)$, but I'm not sure it holds.

- The axiom that nothing can be both an instance and a class does not need to be expressed, since we do not have *inst* relation in our system. We could instead require that nothing can be both a class and a relation, which can be expressed as:

$$error() \leftarrow \exists X(db\_class(X) \wedge db\_relation(X))$$

This can be rewritten into Prolog as:

```
error(both_class_and_instance(X)) :-
    db_relation(X), db_class(X).
```

3.4.6 Inference

The explicit information present in the pancreas diagram is not enough for our knowledge base to work correctly. We can define relation r in the following way:

$$\forall R(\forall X(\forall Y(r(X, R, Y) \leftarrow db\_r(X, R, Y))))$$

In other words, X is related to Y by relation R if such information is explicitly given in the diagram. However, we also need to define how to infer reflexive relationships:

$$\forall R(\forall X(\forall Y(r(X, R, Y) \leftarrow reflexive(R) \wedge r\_reflexive(X, R, Y))))$$

This says that we can use $r\_reflexive$ to see whether reflexive behaviour can be inferred, but only if the given relation is defined to be reflexive. $r\_reflexive$ is defined in the following way:

$$\forall R(\forall X(r\_reflexive(X, R, X)))$$

Transitivity is defined in the following way:

$$\forall R(\forall X(\forall Y(r(X, R, Y) \leftarrow transitive(R) \wedge r\_tr(X, R, Y))))$$

This says that we can use r_tr to see whether transitive behaviour can be inferred, but only if the given relation is defined to be transitive. r_tr is defined in the following way:

$$\begin{array}{lll} \forall R(\forall X(\forall Y(\forall Z(r\_tr(X, R, Z) & \leftarrow & db\_r(X, R, Y) \wedge db\_r(Y, R, Z))))) \\ \forall R(\forall X(\forall Y(\forall Z(r\_tr(X, R, Z) & \leftarrow & db\_r(X, R, Y) \wedge r\_tr(Y, R, Z))))) \end{array}$$

However, the Prolog implementation is slightly different for performance reasons.

We would also like to add inference of properties. We know that

$$r(acinar\_cell, isa, exocrine\_pancreatic\_cell)$$

Hence *acinar_cell* should have all properties that *exocrine_pancreatic_cell* has. This is handled by:

$$\forall R(\forall X(\forall Y(\forall Z(r(X, R, Y) \leftarrow inherited(R) \wedge r(X, isa, Z) \wedge diff(X, Z) \wedge r(Z, R, Y))))))$$

So now we can infer:

```
?- r(acinar_cell,partof,C).
C = exocrine_pancreas ;
C = pancreas ;
No
```

Another interesting question is: since $r(PP\_cell, partof, pancreas)$ and we know that $r(PP\_cell, secrete, pancreatic\_polypeptide)$, should we conclude that $r(pancreas, secrete, pancreatic\_polypeptide)$? In other words, would the relation of secretion be copied up the partonomy? The answer is probably dependent on the view of what it really means to secrete something. If the secreted substance was only used internally within pancreas, we might not want to say that pancreas secretes it, even though some part of pancreas does.

3.4.7 Prolog querying

If we are happy with the closed world assumption, we can use Prolog for querying our knowledge base using simple language resembling English. The context–free grammar of the language is presented below:

$$
\begin{aligned}
\mathit{query} &::= \text{"}\mathit{exist}\text{"}\ \mathit{class}\ \text{"}\mathit{which}\text{"}\ \mathit{relation}\ \mathit{class} \\
&\mid \text{"}\mathit{every}\text{"}\ \mathit{class}\ \mathit{relation}\ \mathit{class} \\
\mathit{relation} &::= \text{"}\mathit{isa}\text{"} \mid \text{"}\mathit{partof}\text{"} \mid \text{"}\mathit{secrete}\text{"} \mid \ldots \\
\mathit{class} &::= \text{"}\mathit{cell}\text{"} \mid \text{"}\mathit{pancreas}\text{"} \mid \text{"}\mathit{capillary}\text{"} \mid \ldots
\end{aligned}
$$

The existential query resembles existential quantification in the sense that one example is enough to get a positive answer. The following Prolog code clarifies how the query is run:

```
ask([exist,Class1,which,Relation,Class2],
    [yes,for,instance,X,Relation,Y]):-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        r(X,isa,Class1),
        r(Y,isa,Class2),
        r(X,Relation,Y).
```

We may be for instance interested to know whether there exists a stem cell, which secretes insulin. In such a case we would ask:

```
?- ask([exist,stem_cell,which,secrete,insulin],A
   ).
No
```

But to see which cells at all secrete insulin, we can ask:

```
?- ask([exist,cell,which,secrete,insulin],A).
A = [yes, for, instance, beta_cell, secrete,
   insulin]
```

Universal querying resembles universal quentification. We can define it in Prolog in the following way:

```
ask([every,Class1,Relation,Class2],[yes,every])
    :-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        \+ counterexample(Class1,Relation,Class2
            ,_).

ask([every,Class1,Relation,Class2],E):-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        counterexample(Class1,Relation,Class2,E)
            .

counterexample(Class1,Relation,Class2,[no,
    because,X,not,Relation,Class2]):-
        r(X,isa,Class1),diff(X,Class1),
        not_r(X,Relation,Class2).

not_r(X,R,Y):- \+ r(X,R,Y).
```

We use negation as failure for finding a couterexample to our universal statement. If a counterexample can be found, the answere is "no" and the counterexample is returned. Otherwise, answer is "yes". A sample interaction with the system may look as follows:

```
?- ask([every,endocrine_pancreatic_cell,secrete,
    insulin],A).
A = [no, because, alpha_cell, not, secrete,
    insulin] ;
A = [no, because, gamma_cell, not, secrete,
    insulin] ;
A = [no, because, pp_cell, not, secrete, insulin
    ] ;
No
?- ask([every,endocrine_pancreatic_cell,partof,
    pancreas],A).
```

```
A = [yes, every] ;
No
```

The full Prolog code is presented in Appendix G on page 257.

Chapter 4

Type systems and programming languages for representing ontologies

This chapter serves as an investigation into programming languages and their usefulness for representing formal ontologies and implementing ontology–based natural language tools. One of the topics we will focus on is the notion of "type". We will explain how this notion is used in logic and ontologies. We will further see how it is utilized in programming languages, where it is a key concept [36]. In other words, we will discuss what types of types there are. We will discuss the ontological usefulness of several programming languages as far as their type systems and other related features are concerned. These considerations are important, as in the current work we are not only interested in finding a link between natural language semantics and formal ontologies, but also in establishing how to approach them in a practical way from the point of view of computation.

4.1 Logical types

In First Order Predicate Logic there is no notion of types, or we can think of it as having only one type, namely that of individuals. However, in the logical type theory types are the key concept. We have two elementary types:

- o – the type of truth values
- ι – the type of entities/objects

Those types are further combined by forming a function type in order to recursively form infinitely many possible types. For any two types τ_1 and τ_2 we can construct a type of functions from τ_1 to τ_2, denoted either $\tau_1 \to \tau_2$ or $\tau_2\tau_1$. We prefer the latter notation, as it is more compact.

For instance, type of the negation operation will be oo, that is negation is a function that takes a truth value and produces another truth value.

Some very useful kinds of objects, e.g. sets can be represented with the help of function types. We could represent the type of sets of individuals by $o\iota$, that is a function that takes an individual as argument and returns a truth value as output. The intended working of such a function is that when an individual is in the set, the function returns true, and false otherwise.[1]

For instance the set

$$\{a, b\}$$

can be represented by the following function:

$$f(X) = \top \text{ if } X = a \text{ or } X = b, \quad \bot \text{ otherwise.}$$

However, this poses some restrictions on the use of such types from the computational point of view. One of the most famous undecidable problems is the problem of equality of functions. There is no algorithm that could decide whether two arbitrary functions (even of the same type) are equal or not.

Consider the following function as an illustration of this problem:

$$f(X) = \top \text{ if } (X = a \wedge \text{`}p{=}np\text{'}) \text{ or } X = b, \quad \bot \text{ otherwise.}$$

In the above we added *'p=np'*, the famous computer science problem as one of the conditions as to what the function will answer when given a as the argument. Here for brevity we simply refer to it by the name *'p=np'*, but think of it as a long formula formalizing the problem and being either true or false. If *'p=np'*, the function will return $\top$, otherwise it will return $\bot$. Hence, this function is identical to the previous one if and only if *'p=np'*. This illustrates that the problem of comparing two functions is at least as difficult as the *'p=np'* problem, i.e. it is not easy to say the least.

Hence, using type theory poses great difficulties for the implementation, as most objects are represented by functions. Therefore, we would not be able to

[1] This commonly used function is known as the characteristic function of a set.

compare various objects (decide whether they are equal or not) in type theory. This is a huge disadvantage, taking computation into consideration.

4.2 Types in programming languages

Types for data in programming languages (sometimes called datatypes) are in fact kinds of ontologies for objects created in computer's operating memory. Some of them enable the representation of large taxonomies via the so–called "inheritance". The main objective behind introduction of types in programming languages is to eliminate many programming mistakes stemming from inter–type confusion. Those mistakes are very common in non–typed languages, but can be easily detected by the compiler for typed languages. Let us now have a brief look at the declarative programming languages.

Declarative languages describe what should be computed, rather than how something should be computed. This means they usually describe the computation without state, side effects, and control flow. The type system of declarative languages is very well suited for expressing already specified logical theories. Some of the advantages of declarative languages include:

- Most declarative languages have a formal or semiformal semantics, which allow for proving the correctness of the program.
- Declarative languages have a syntax which is very close to the standard mathematical notation. It is therefore easy to express formal ideas as a program.
- Because a declarative program describes *what* the solution is instead of *how* to compute it, the source code for the program is around 10 times more concise than one expressed in a imperative language.
- Strongly–typed declarative languages tend to have stronger type systems than strongly–typed imperative languages. Consider e.g. the Java imperative language, where `null` is a member of any user–defined type, or C/C++, where compiler automatically casts between most primitive integer types, pointers, etc. While those features are often very useful for programmers, they also represent weakenings of the type system itself.
- Some declarative languages have support for nondeterministic computation (returning multiple results), which at times proves to be very helpful.

It is however worth noting that the type system of a programming language cannot be used easily for representing ontologies. The ontologies are usually represented in a program as data, not as types. With this approach, ontological types will take form of dynamic self–programmed types, which are taken care of by the program and not by the compiler or type checker.

Let us consider an example. In Java, a very popular language with object–oriented type system, we could declare the class of substances:

```
class Substance {}
```

Then, we could create a subclass of it, expressing that insulin is one of subtypes of substance:

```
class Insulin extends Substance {}
```

Java even provides a so–called reflection mechanism, where a program can look into the type system during runtime. However, this approach is only suitable for expressing fixed, bounded–size ontologies. The generative ontologies that we are going to work with are of unbounded size, as we can create complex classes during the process of analyzing the text. We need to dynamically create new ontological types as we go, and therefore we can't use the programming language's type system directly for representing the ontology.

Next, let us have a brief look at some of the declarative programming languages, together with their type system.

4.2.1 StandardML

StandardML is an established functional programming language. It is widely used in academia, as it has a formal semantics. The following description [29] of StandardML [31] is a particularly concise description of the language:

> *Standard ML is a safe, modular, strict, functional, polymorphic programming language with compile–time type checking and type inference, garbage collection, exception handling, immutable data types and updatable references, abstract data types, and parametric modules. It has efficient implementations and a formal definition with a proof of soundness.*

Unfortunately, one can stumble upon an obstacle along the way. It turns out that some of the operations which are necessary for natural language handling

would benefit greatly from nondeterministic implementation. Unfortunately, StandardML does not provide language support for nondeterminism.

Consider, e.g. the problem of recognizing the ontological relations, based on prepositions. Let us assume that we would like to have the following sample mappings:

Preposition	Relation	Example
on	TMP	on Monday
to	DST	to school
in	TMP	in winter
in	LOC	in cells

As we can see, preposition "in" can describe two different relations, temporal placement and locational placement. Unfortunately, one cannot write in StandardML:

```
datatype role
  = TMP (* temporal aspects (generic role) *)
  | LOC (* location, position *)
  | DST (* destination of moving process *)

fun p2r "on" = TMP
  | p2r "to" = DST
  | p2r "in" = TMP
  | p2r "in" = LOC
```

This program will not compile, because of the way pattern matching works in functional languages. Only the first matching definition is used, so the pattern

```
  | p2r "in" = LOC
```

is unreachable.

Of course workarounds exist for that problem, e.g. the function p2r instead of having the type `fn : string -> role`, could have type `fn : string -> role list`, so instead of nondeterministically returning a role, it could return deterministically a list of roles:

```
datatype role
  = TMP (* temporal aspects (generic role) *)
  | LOC (* location, position *)
  | DST (* destination of moving process *)
```

```
fun p2r "on" = [TMP]
  | p2r "to" = [DST]
  | p2r "in" = [TMP,LOC]
```

In this way, one could implement a behaviour that is a substitute for nondeterminism, using exception throwing for backtracking. It might be feasible to follow that path, however at some point the code can became very convoluted, especially if nondeterminism is used in more than one place. It could lead to the use of lists of lists of lists.

Therefore, it might be beneficial to turn to a language with support for nondeterminism.

4.2.2 Prolog

Prolog is the most popular programming language supporting nondeterminism. It is based on SLD–resolution and unification with negation as failure and a closed world assumption. It is used extensively in the literature on computational natural language semantics. [10, 35]

Continuing our example from Section 4.2.1, we can simply write:

```
p2r(on,tmp).
p2r(to,dst).
p2r(in,tmp).
p2r(in,loc).
```

If we now run the program with the goal ←`p2r(in,X)`, we will get two answers for `X`, i.e. `X=tmp` and `X=loc`.

This is a great feature allowing very easy formulation of the problem at hand. Unfortunately, Prolog does not come with a type system. The programmer needs to verify by hand that the terms used will unify iff they are supposed to. In particular, the heavy use of lists introduces some problems.

Once the programmer makes a type mistake (e.g. term has the arity of three, instead of four), Prolog cannot spot it. In such a case the only thing that happens is that Prolog is saying `NO` and not giving any answers, because the unification fails at some stage.

Since this sort of behaviour is not considered erroneous by Prolog, the only way of detecting such a simple typing error is tracing the execution of the program step–by–step until one realizes where the problem lies. One can spend

countless hours on such unnecessary debugging, and therefore it might be advisable to use a language with embedded type system.

4.2.3 Mercury

The following description [44] of Mercury is a concise description of the language:

> *Mercury is a new logic/functional programming language, which combines the clarity and expressiveness of declarative programming with advanced static analysis and error detection features. Its highly optimized execution algorithm delivers efficiency far in excess of existing logic programming systems, and close to conventional programming systems. Mercury addresses the problems of large-scale program development, allowing modularity, separate compilation, and numerous optimization/time trade-offs.*

Type system

The most important feature of Mercury is its strong type system. The type system is very similar to the one known from functional languages, like Haskell or ML. It includes discriminating union, recursive and polymorphic types. A type definition for a well–known append predicate has the following form:

```
1 :- pred append(list(T), list(T), list(T)).
```

In the above T is a type variable, and stands for any type, including lists, lists of lists, functions, etc.

Determinism declaration

Mercury allows for creating the following kinds of predicates and functions:

- Deterministic – predicate can succeed exactly once.
- Semi–deterministic – predicate can succeed at most once.
- Multi – predicate always succeeds more than once.
- Nondeterministic – predicate can succeed any number of times.
- Erroneous – predicate cannot succeed.

In addition there are “commited choice” nondeterministic predicates.

The determinism of a predicate has to be defined by the programmer and is strictly checked by the compiler. The compiler can infer the correct nondeterminism declaration for most predicates.

If the program is not written clearly, the compiler may overestimate the behaviour of a predicate, e.g. it can infer that a predicate is nondeterministic, while in fact it is semi–deterministic. This can usually be easily fixed by a small re–write of the code. Usually one should use more functional programming style than logic programming style for semi–deterministic and deterministic predicates. For instance, using if–then–else constructs helps a great deal, as in such a case compiler knows that either “then” branch will be taken, or “else”, but not both, and not neither.

While the necessity of declaring the determinism of a predicate might seem like a waste of time, it is not. There are certain programming errors that are detected by the compiler in this way.

The knowledge that compiler gains from the determinism declarations allows it to generate very efficient code. Deterministic and semi–deterministic predicates, for instance, do not have a need for backtracking. Mercury programs tend to work around 10–100 times faster than SWI–Prolog programs [37].

Mode system

The Mercury language has the notion of instantiatedness of a variable. It corresponds to a specific instantiation of the type constructors, which form a value. Consider the following simple example:

```
:-inst non_empty_list
        == bound([ground|ground]).
```

Here we can see that the value is `bound` to the list constructor (`[|]`) and both head and tail are `ground`. Hence, the list must be non–empty. Two most common instantiatednesses are `ground`, where the term does not contain any variables, and `free`, meaning that variable is not instantiated at all.

Those definitions are in turn used in so–called modes. The mode of a predicate declares what happens with the parameters of a predicate during its execution. Two most common modes are:

```
:- mode in == ground >> ground.
:- mode out == free >> ground.
```

In other words, a variable is an input to a predicate, if its `ground` both when the predicate is called, and when its finished. A variable is an output of a predicate, if its `free` when the predicate is called, and it is `ground` when its finished.

The mode system is immensely helpful for avoiding common mistakes in logic programming. Consider for instance the well-known list appending predicate, where type, determinism, and mode declarations look as follows:

```
:- pred append(list(T), list(T), list(T)).
:- mode append(in, in, out) is det.
:- mode append(out, out, in) is nondet.
```

One needs to declare that all three arguments are lists of the same type of elements. Additionally, we want to use `append` in two ways. First, given two input lists, we want to concatenate them to deterministically produce a resulting list. Secondly, given a list, we want to nondeterministically produce two lists, which when concatenated, give us the supplied list.

Apart from serving as a nice overview of how the predicate works, such a declaration allows the compiler to check for the following sample mistakes:

- Type errors – if the predicate tries to unify one of the arguments with a list of something else than the other arguments.
- Determinism errors – if the mode which we declared to work deterministically, is in fact nondeterministic.
- Mode errors – If the `out` variable is not instantiated to be ground by the predicate.

Pure declarativeness

In most declarative languages there is a need for impure functions or predicates which introduce side–effects. Side–effects are things that happen outside of the formal semantics of a language. This is very often the case with e.g. input/output operations. In Mercury input and output is performed with preservation of purely declarative semantics. This can only be achieved thanks to the mode declarations for predicates. Consider, e.g. the predicate for loading WordNet database:

```
:-pred load_wn(string::in,
               wn::out,
```

```
                io::di,
                io::uo) is det.
```

The two last parameters represent input/output state before and after the predicate was called. They have special modes. `di` means "destructive input" – the variable is dead after being passed to `load_wn`, cannot be used again. `uo` means "unique output" – the variable can be used only once after being output from the predicate. Please observe that the predicate cannot be nondeterministic, as it is impossible to backtrack over input/output operations. One cannot "undisplay text" or "unread from a file". This is different than in Prolog, where input/output is performed in impure way, so one can perform input/output operations in backtracking predicates.

4.3 Ontological types

Having discussed some datatypes in some programming languages, we now return to the issue of representing ontologies. From the point of view of the ontologies, types take form of concepts or classes in the ontology. This means that the taxonomy of the domain in question defines what types of objects can exist in the domain.

Accordingly, the ontological types can be defined as a simple lattice, where nodes represent classes. But we can also use some more heavy–weight types, where an inference engine would need to be employed for deciding whether some object belongs to a given type. This can be achieved by using Description Logics with a large and complicated TBox.

If we wish to base our ontological types on the pancreas diagram from Figure 3.2 on page 60, then the snapshot of such a TBox could look as presented in Figure 4.1 on page 79.

Notice that since in the biomedical domain we do not concern ourselves with individuals, there is no need for the ABox.

In programming languages we usually have a finite number of types in a given program. But from ontological perspective, types could be defined in a generative fashion, hence producing potentially infinitely many (unbounded number of) available classes. This approach is followed in our ontological semantics prototypes with the Peirce product and infinite generative ontologies defined by means of conceptual feature structures.

$$
\begin{aligned}
pancreatic\_cell &\sqsubseteq cell \sqcap \forall located\_in.pancreas \\
exocrine\_cell &\sqsubseteq cell \sqcap \forall secretes.enzyme \\
endocrine\_cell &\sqsubseteq cell \sqcap \forall secretes.hormone \\
exocrine\_pancreas &\sqsubseteq \forall partof.pancreas \\
endocrine\_pancreas &\sqsubseteq \forall partof.pancreas \\
stem\_cell &\sqsubseteq cell \\
adult\_stem\_cell &\sqsubseteq stem\_cell \\
embrionic\_stem\_cell &\sqsubseteq stem\_cell \\
islet\_of\_Langerhans &\sqsubseteq \forall partof.endocrine\_pancreas \\
exocrine\_pancreatic\_cell &\sqsubseteq exocrine\_cell \sqcap pancreatic\_cell \\
&\quad \sqcap \forall partof.exocrine\_pancreas \\
endocrine\_pancreatic\_cell &\sqsubseteq endocrine\_cell \sqcap pancreatic\_cell \\
&\quad \sqcap \forall partof.islet\_of\_Langerhans \\
duct &\sqsubseteq \forall partof.exocrine\_pancreas \\
capilary &\sqsubseteq \forall partof.islet\_of\_Langerhans \\
\alpha\_cell &\sqsubseteq endocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.glucagon \\
\beta\_cell &\sqsubseteq endocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.insulin \\
\delta\_cell &\sqsubseteq endocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.(gastrin \sqcup somatostatin) \\
PP\_cell &\sqsubseteq endocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.pancreatic\_polypeptide \\
centroacinar\_cell &\sqsubseteq exocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.digestive\_enzyme \\
&\quad \sqcap \forall has\_primary\_signal.secretin \\
acinar\_cell &\sqsubseteq exocrine\_pancreatic\_cell \\
&\quad \sqcap \forall secretes.bicarbonate\_ion \\
&\quad \sqcap \forall has\_primary\_signal.cck
\end{aligned}
$$

Figure 4.1: Pancreas ontology translated into Description Logics.

4.4 Grammatical ontotypes

It is possible to use the familiar notion of context–free grammars for representing ontological types, as described in [1]. In such an application classes are represented by nonterminal symbols and class subsumption relationship is replaced by the string derivation.

As an example, consider the following sample grammar, which represents an ontology created by combining Basic Formal Ontology with Gene Ontology. The grammar is presented in Figure 4.2 on page 81.

This grammar has the power of generating infinitely many ontological types by means of deriving many various sentential forms. A sample derivation is shown in Figure 4.3 on page 82.

$$
\begin{array}{rcl}
\langle \text{Entity} \rangle & ::= & \langle \text{Continuant} \rangle \\
& | & \langle \text{Occurrent} \rangle \\
\\
\langle \text{Occurrent} \rangle & ::= & \langle \text{Occurrent} \rangle \langle \text{Occurrent role list} \rangle \\
\\
\langle \text{Occurrent role list} \rangle & ::= & \epsilon \\
& | & [\langle \text{Occurrent role} \rangle] \langle \text{Occurrent role list} \rangle \\
\\
\langle \text{Occurrent role} \rangle & ::= & CBY : \langle \text{Occurrent} \rangle \\
& | & LOC : \langle \text{Continuant} \rangle \\
& | & WRT : \langle \text{Entity} \rangle \\
& | & BMO : \langle \text{Entity} \rangle \\
& | & POF : \langle \text{Occurrent} \rangle \\
& | & \ldots \\
\\
\langle \text{Continuant} \rangle & ::= & \langle \text{Continuant} \rangle \langle \text{Continuant role list} \rangle \\
\\
\langle \text{Continuant role list} \rangle & ::= & \epsilon \\
& | & [\langle \text{Continuant role} \rangle] \langle \text{Continuant role list} \rangle \\
\\
\langle \text{Continuant role} \rangle & ::= & LOC : \langle \text{Continuant} \rangle \\
& | & POF : \langle \text{Continuant} \rangle \\
\\
\langle \text{Occurrent} \rangle & ::= & \langle \text{ProcessualEntity} \rangle | \ldots \\
\langle \text{ProcessualEntity} \rangle & ::= & \langle \text{Process} \rangle | \ldots \\
\langle \text{Process} \rangle & ::= & \langle \text{biological\_process} \rangle \\
\\
\langle \text{Continuant} \rangle & ::= & \ldots
\end{array}
$$

Figure 4.2: Grammatical specification of a sample ontology.

$$
\begin{array}{c}
\langle \text{Entity} \rangle \\
\Downarrow \\
\langle \text{Occurrent} \rangle \\
\Downarrow \\
\langle \text{Occurrent} \rangle \langle \text{Occurrent role list} \rangle \\
\Downarrow \\
\langle \text{Occurrent} \rangle [\langle \text{Occurrent role} \rangle] \langle \text{Occurrent role list} \rangle \\
\Downarrow \\
\langle \text{Occurrent} \rangle [\langle \text{Occurrent role} \rangle] \\
\Downarrow \\
\langle \text{Occurrent} \rangle [PNT : \langle \text{Continuant} \rangle] \\
\Downarrow \\
\langle \text{ProcessualEntity} \rangle [PNT : \langle \text{Continuant} \rangle] \\
\Downarrow \\
\langle \text{Process} \rangle [PNT : \langle \text{Continuant} \rangle] \\
\Downarrow \\
\langle \text{biological\_process} \rangle [PNT : \langle \text{Continuant} \rangle] \\
\Downarrow \\
\langle \text{transport} \rangle [PNT : \langle \text{Continuant} \rangle] \\
\Downarrow \\
\vdots \\
\Downarrow \\
\langle \text{transport} \rangle [PNT : \langle \text{Substance} \rangle] \\
\Downarrow \\
\vdots \\
\Downarrow \\
\langle \text{transport} \rangle [PNT : \langle \text{glucose} \rangle]
\end{array}
$$

Figure 4.3: Sample sentencial form derivation for constructing nested ontological types

Chapter 5

Ontological Semantics

In this chapter we will have a closer look at natural language, in particular English. In particular we will discuss its grammar from a formal perspective. We will also try to establish how to formally represent the meaning of sentences and phrases with the help of generative ontologies. Further, we will discuss some problems related to the extraction of that meaning.

Ontological phenomena visible most clearly in generative ontologies are mirrored in the language. Our observation is that ontology should be language–independent, while language is perhaps ontology–dependent to a large extent. Hence, it is quite natural to base ontological semantics for a language on generative ontologies.

5.1 Meaning of sentences

Let us use First Order Predicate Logic as a metalogic for describing relation between sentences and their meaning. We will write:

$$follows(\Phi, \Upsilon)$$

whenever Φ follows from sentence Υ. Υ is a natural language sentence represented in arbitrary form, e.g. as a list of words. Φ is a formula in some chosen logic representing partial meaning of the sentence Υ. We will say:

$$means(\Upsilon, \Phi)$$

whenever formula Φ is the full meaning of the sententence Υ. Let us define what we understand by "full meaning":

$$\forall S, M(means(S, M) \leftrightarrow follows(M, S) \wedge \forall I(follows(I, S) \rightarrow entails(M, I)))$$

In other words formula M is the full meaning of sentence S if it follows from S and if it entails all other formulas I, which also follow from S. So when $means(\Upsilon, \Phi)$, Φ is the perfect translation of natural language sentence Υ into desirable logic, where no information is lost, and of course no new information is introduced.

5.2 Automatic meaning extraction

When constructing a system for automatic meaning extraction, one cannot hope to extract the full meaning of each sentence. There are several reasons for that:

- The reasoning necessary for full understanding is beyond computer's abilities. It may be necessary to reference complicated real world knowledge to understand the sentence.
- The target logic is unable to represent the full meaning of a sentence due to its limited expressivity.

In any case, the system for natural language semantics extraction is supposed to understand as much of the sentence as possible. We are interested in extracting a meaning Φ, such that:

$$reading(\Upsilon, \Psi) \wedge entails(\Psi, \Phi)$$

5.3 Human language understanding

Human beings are exceptionally good at understanding language. It is by far the most unique way of communicating found in nature, as it is tied to only one species.

Since our goal is to allow computer to understand language, we shall first discuss what makes language understandable for us. Indeed, if we knew how our mind grasps the language, we could try to write a program, which would act in a similar manner.

Unfortunately, this task might be extremely difficult, or maybe even impossible. As a matter of fact we can only understand the brain on two levels:

1. A very low level: we know that impulses are being sent between cells and we can observe that some regions of the brain are active while performing a particular task, such as speaking.

2. A very high level: we can observe the external behavior produced by our mind.

The first understanding can be compared to how computer hardware functions. The second is similar to the output of a computer program, or its interface. We do not have any access to the "source code" for our mind.

5.4 Language as a protocol

When we want two machines to communicate, we need a protocol, which both will follow, so that they can understand each other. For human beings, the natural language serves as a protocol.

The language, which we use today, and which has been evolving for tens of thousands of years has one particular feature, which distinguishes it from other protocols used today, i.e. it is extremely complicated. This stems from the fact that language evolved to serve purposes of human mind, not that of a computer.

This might be conceived as an advantage in a various contexts, as the ability of describing the same concept in so many different ways allows us to express our emotions, it makes reading novels so pleasurable, etc.

Unfortunately, from the software engineering perspective the complexity of language is a huge problem. No software system is capable of understanding it fully (or even to a large extent).

Luckily, language being a protocol, is not completely random, but rather sentences are built in a specific way. Language has been studied by linguists and the knowledge about it is extremely useful when one tries to design a software system, which should take natural language sentences as the input.

In Sections 5.4.1 and 5.5 we explain the particularly interesting features of natural language, which might make it computer–understandable to a certain degree.

5.4.1 Parts of speech

The English language consists of words, which can be grouped into certain classes, called parts of speech or lexical categories. The most common ones are:

- common noun – names object class, e.g. the noun "cell" has the meaning, which covers all imaginable cells, not a particular cell.
- proper noun – names particular object, e.g. "Denmark" is a proper noun referring to a particular country.
- verb – names actions, which are activities or interactions between things, e.g. "stored" is a verb, which names the action of storing something somewhere.
- adjective – names properties ascribed to nouns, e.g. the phrase "hyperglycemic symptoms" ascribes the property of being hyperglycemic to symptoms.
- preposition – explains how different entities are related to each other. Very few prepositions exist. They tend to capture only the most common relations. E.g. the phrase "storage in cell" means that storage takes place inside of a cell.
- adverb – plays similar role as adjective, except that it describes properties of concepts represented by other parts of speech than nouns.

5.4.2 Necessity of part–of–speech tagging

Let us consider the following sentence:

insulin forces storage

Let us assume that the underlying ontology understands that the word "insulin" represents the concept c_insulin and the word "storage" represents the concept c_storing. c_insulin is defined as a chemical substance called insulin. c_storing is defined as an action of storing something somewhere. Let us also assume that nothing in our ontology corresponds to the word "forces".

In such a case understanding the whole phrase could be attempted by skipping the word "forces" and analyzing just "insulin storage". Unfortunately, this would result in an incorrect understanding, as the original phrase was not really talking about insulin storage. It is not insulin that is in fact stored.

In order to remedy the situation, the algorithm could use part–of–speech tagging. This process assigns a list of lexical categories to each word of the sentence. The coverage of part–of–speech tagging is extensive, due to the existence of large lexical resources, like WordNet. The size of todays lexical resources is far superior to the size of any existing ontology. Almost any word of English

language can be assigned a lexical category, but only a small subset can be found in any ontology.

After the application of the part–of–speech tagging, we are faced with the following information:

sentence	insulin	forces	storage
tagging	noun	verb	noun
ontology	c_insulin	?	c_storing

At the moment the algorithm understands, that the sentence does not talk about storage of insulin, as words "insulin" and "storage" appear on two sides of a verb. Such a situation disallows the compound concept to be understood as storage of insulin. Rather, insulin could be an agent and storage could be a patient of some action, described by the verb.

Observe that if the middle word is, e.g. a noun, we might be faced with a different situation. For instance: "insulin particles storage" sentence has a different structure, where the second word describes a patient for the action described by third word. In other words, we deal with the concept of storing some particles.

5.5 Grammar as the structure of English

5.5.1 Rules and structure

All sentences of a language are similar. Even though completely different words are used, the structure of the sentence follows some rules. Recognizing those rules is of utmost importance for understanding the sentence. If the sentence is not constructed according to the rules, it will not be understood or it will be misunderstood.

With artificial languages (e.g. programming languages), knowing the rules is simple. When language is designed, a set of rules is invented, usually using a variant of Backus–Naur Form if the language is context–free. One might therefore construct all the valid sentences of the language with the rules provided.

5.5.2 Rules for natural languages

The problem with natural languages is that they were not designed with any rules in mind. The language evolved together with human brain, and we somehow know from the very childhood, which sentences are correct and which are not (to some extent). We learn that by listening to sample correct sentences, rather than learning some rules and following them.

Hence, we are faced with a problem, which appears only for natural languages. The problem is that we do not know the rules of the language. In a sence the rules are hidden.

Why would we need the rules at all? It is because we would like to teach computers to understand natural language. Unfortunately, inputting all valid sentences into a database is not an option. Such a set would probably be infinite, or in most optimistic estimation extremely huge. So we shall try to construct a relatively small set of rules, which could describe the structure of the language in a satisfactory way.

In order to learn the rules, we might assume that we know which sentences are correct and which are incorrect, and based on that we would like to analyze the language, so that we can recognize the hidden rules.

Such a task, however, turns out to be extremely difficult.

A substantial problem is that whatever set of rules one might come up with, it is always relatively simple to find a counterexample. Such a counterexample could take two forms. It could either be a valid sentence, which cannot be produced using the rules, or it could be an invalid sentence, which could be produced using the set of rules provided.

For this very reason one cannot prove that English is a context–free language, hence it might not be describable using a context–free grammar.

Many linguistic problems that are faced if one tries to construct a context–free grammar for English are described in [40].

5.5.3 Shallow context–free grammar for English

This section describes the idea of a shallow grammar for English language.

The problems described in Section 5.5.2 have to be dealt with somehow if we want the system to understand the language. Fortunately, the system, which is to be designed in the current text, has some important properties. The crucial thing to realize is that the system has to analyze biomedical documents. Such documents are written using correct English language. Therefore, we can assume the absence of incorrect sentences in the analyzed text.

Such assumption has a major influence on the set of rules, which we shall decide upon. Normally the rules should be very restrictive, so that they need to reject incorrect sentences. However, since we will not have any incorrect sentences, we do not need to reject them with the help of our rules.

As an example, consider the following simple rule[1]:

[1] Expressed in BNF.

$$\langle \text{sentence} \rangle ::= \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{noun} \rangle \tag{5.1}$$

This rule expresses that a sentence can consist of a noun, followed by a verb, followed by a noun. Let us consider the following two sentences:

1. Insulin forces storage.

2. Insulin force storage.

Sentence 1 is a correct English sentence, while sentence 2 is not. Nevertheless, sentence 1 will be accepted by our rule, as will sentence 2.

If we wanted to reject the second sentence, but accept the first one, we would need to introduce more complicated rules. We can however observe, that the second sentence will not ever appear in the analyzed text, assuming that it is a high–quality material.

Hence, we can use a relatively simple shallow grammar for analyzing English.

As previously noted, English might not be describable by a context–free grammar. However, taking into consideration the aforementioned observation, we can try to formulate a shallow grammar as a context–free grammar, since we do not need to reject incorrect sentences.

Such a grammar would not cover the whole English language, but this is not a problem. We shall note that the text, which is to be analyzed, is a scientific text. As such it normally makes use of a specific subset of English language only.

As an example, we could consider the problem of questions. In a casual conversation, a novel, or a play, questions occur very often. This is however not a case for scientific articles, reports and texts. The Wikipedia entry for "Insulin", which is our playground, does not contain even one question. That observation allows us to construct a substantially simpler grammar for text analysis, as the order of words in a question is not the same as in a statement.

In case a question appears in the analyzed text, we simply will not be able to analyze such a sentence. Similarly if a sentence formulated in a less obvious way is encountered, which is rejected by the grammar, it will not be analyzed.

Taking all the considerations into account, we might design the a shallow context–free grammar for English, which is presented in Figure 5.5.3 on page 90.

Please observe that the grammar is ambiguous. The same sentence could produce different parse trees.

Figure 5.1: A shallow context-free grammar for English, specified in BNF.

$$
\begin{array}{rcl}
\langle \text{s} \rangle & ::= & \langle \text{d\_a\_cnp\_pp} \rangle \langle \text{vp} \rangle \\
\langle \text{vp} \rangle & ::= & \langle \text{v} \rangle \langle \text{pps} \rangle | \langle \text{v} \rangle \langle \text{d\_a\_cnp\_pp} \rangle \langle \text{pps} \rangle \\
\langle \text{pps} \rangle & ::= & \epsilon | \langle \text{pp} \rangle \langle \text{pps} \rangle \\
\langle \text{pp} \rangle & ::= & \langle \text{p} \rangle \langle \text{d\_a\_cnp\_pp} \rangle \\
\langle \text{d\_a\_cnp\_pp} \rangle & ::= & \langle \text{d\_a\_cnp} \rangle \langle \text{pps} \rangle \\
\langle \text{d\_a\_cnp} \rangle & ::= & \langle \text{d} \rangle \langle \text{a\_cnp} \rangle | \langle \text{a\_cnp} \rangle \\
\langle \text{a\_cnp} \rangle & ::= & \langle \text{adj} \rangle \langle \text{a\_cnp} \rangle | \langle \text{cnp} \rangle \\
\langle \text{cnp} \rangle & ::= & \langle \text{n} \rangle \langle \text{cnp} \rangle | \langle \text{n} \rangle \\
\langle \text{v} \rangle & ::= & \mathit{force} | \mathit{forces} | \mathit{cause} | \mathit{causes} | \ldots \\
\langle \text{n} \rangle & ::= & \mathit{insulin} | \mathit{storage} | \mathit{cell} | \mathit{cells} | \ldots \\
\langle \text{d} \rangle & ::= & \mathit{the} | \mathit{this} | \mathit{my} | \ldots \\
\langle \text{p} \rangle & ::= & \mathit{in} | \mathit{with} | \mathit{by} | \ldots \\
\langle \text{adj} \rangle & ::= & \mathit{pancreatic} | \mathit{recombinant} | \mathit{adipose} | \mathit{pineal} | \ldots
\end{array}
$$

5.6 Let us try various formalisms

Instead of giving one final answer as to what formalism shall be used for representing ontological semantics, we would prefer to "play around" with some of them.

Initially one can try each of them separately and argue how well suited each of them is for representing ontological semantics. Is Description Logics good enough? How about First Order Predicate Logic? Lambda Calculus? The problems of tractability and decidability have to be taken into consideration as far as search aspects are concerned, so the reader knows that semantics expressed in λ–calculus or First Order Predicate Logic cannot be searched for. Hence a need arises for narrowing or combining some of them.

At this point let us have a look at the Peirce product[2] based semantics for representing nodes in generative ontology, which is restrictive enough to be tractable.

5.7 Peirce–algebraic ontological semantics

We are interested in "understanding" the language. This is a very complicated problem, and in the computational context it can be defined as capturing correct semantics from the text.

There are various theories describing formal semantics for English, e.g. the algebraic boolean semantics described in [26].

However, the most prominent example is the Montague semantics, where one focuses on translating various English phrases into quantified formulae in First Order Predicate Logic[3].

As an example, consider the following article from the Wikipedia entry on "Insulin":

"Insulin is required for all animal life." (5.2)

One could capture the semantics of this sentence in the following way:

$$\forall X(life(X) \wedge animal(X) \rightarrow requires(X, insulin))$$

This semantic representation is not particularly useful for the purpose of creating a search engine. One of the problems is that First Order Predicate

[2]The Peirce product has been introduced in section 2.3.7 on page 47.
[3]Often higher–order logics are also used.

Logic is not decidable. Therefore it would be impossible to decide whether semantics of some sentence in the text subsumes the semantics of the query sentence.

Additionally, such a focus on quantification is not something that the user of the search engine would expect. As a matter of fact the user is mostly concerned with finding articles, which refer to a particular concept. Here the concept could be defined as requiring of insulin by animal life.

Peirce algebra (described in Section 2.3.7 on page 47) turns out to contain a very useful operation, called Peirce product, which allows us to express the ontological semantics of the sentence of interest as:

$$requiring \cap agt : insulin \cap pnt : (animal \cap life)$$

Note that Peirce product (:) binds stronger than intersection ($\cap$).

In order to explain this ontological semantics, let us go back to our running example:

$$\textit{Insulin forces storage of glycogen.} \tag{5.3}$$

Its semantics can be expressed using Peirce algebra as:

$$forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \tag{5.4}$$

Recall that Peirce algebra is an algebra of sets and relations between individuals belonging to those sets [12]. In Equation 5.4, the sets, which represent classes in our ontology, are:

- $forcing$ – A class (set) comprising all imaginable actions of forcing. E.g. "me forcing my younger brother yesterday to clean the room" is an example of individual belonging to this class.
- $insulin$ – This class contains only one individual, being the chemical substance called "insulin". It is a singleton set.
- $storage$ – A class (set) comprising all imaginable actions of storing.
- $glycogen$– This class contains only one individual, being the chemical substance called "glycogen. It is a singleton set.

Similarly, the following ontological relations are present in Equation 5.4:

- agt – Agent relation, relating an action to its agent, which is usually initiating the action and carrying it on.

- *pnt* – Patient relation, relating an action to its patient, which is usually influenced by the action, being modified, etc.

The problem of capturing the semantics presented in Equation 5.4 from the sample sentence is described in Chapter 7 on page 113.

In order to fully understand the semantics at hand, let us first translate it according to the definition of Peirce product:

$$
\begin{aligned}
& forcing \\
\cap \quad & \{X | \exists Y((X,Y) \in agt \ \wedge \ Y \in insulin)\} \\
\cap \quad & \{X | \exists Y((X,Y) \in pnt \\
& \qquad \wedge \ Y \in (storage \cap \{X | \exists Y((X,Y) \in pnt \ \wedge \ Y \in glycogen)\}))\}
\end{aligned}
$$

The resulting formula is quite complex. Let us try to explain part of it:

$$\{X | \exists Y((X,Y) \in agt \ \wedge \ Y \in insulin)\} \tag{5.5}$$

Equation 5.5 defines a set of all individuals X, which are agent–related to some individual Y, such that Y is an instance of *insulin*. The resulting set will be a class, comprising all entities, which have insulin as an agent.

Coming back to the full sentence, if we intersect all imaginable forcing actions with all entities having insulin as an agent, we get a class containing all forcing actions, which have insulin as an agent.

Similarly, we narrow the $forcing$ class by intersecting it with all entities, which have storage as a patient. A further, Peirce product restriction is applied to the *storage* in a nested way, as we are only interested in the storage of glycogen.

Hence, by using Peirce algebraic operations we can construct infinitely many compound concepts, using a finite supply of primitive classes and relations. It is worth noticing that such a representation allows us to capture most of the substantial information from the text.

This semantics is chosen as the best alternative for the ontological natural language analyzers, which are described in Chapter 7.

A very useful property of the chosen semantics is that it is variable–free, hence quite simple to be dealt with computationally. It does not suffer from the problems of First Order Predicate Logic, like undecidability. It is used in the working implementation of the system, and allows it to work very efficiently.

5.8 Semantic incompleteness

If we consider sentence 5.3 and its semantics 5.4, we can notice that not all of the information that humans can retrieve from the sentence is captured by the semantics. For instance, the tense of the verb is lost, so the semantics does not tell whether the action of forcing happens presently, in the past, etc.

In other words, we have:

$$\begin{aligned} & [\![\text{Insulin forces storage of glycogen.}]\!] \\ = \; & [\![\text{Insulin forced storage of glycogen.}]\!] \\ = \; & forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \end{aligned}$$

We should discuss whether such a behaviour is desired. We have to keep in mind that the semantics presented is supposed to be used by a search engine for ontological querying.

Clearly, the search engine will give one of the sentences as the match for the other, as they have the same semantics. This is a desired behaviour, because the sentences talk about the same concepts, and we would probably like to neglect the tense when we use the search engine.

On the other hand, we need to keep in mind that such a phenomenon cannot be used as the main principle for the search engine's matching. Also sentences with different semantics should be often found as answers to queries. Consider e.g.:

$$\begin{aligned} & [\![\text{Insulin forces storage of glycogen.}]\!] \\ = \; & forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \end{aligned}$$

and:

$$\begin{aligned} & [\![\text{Insulin causes storage of glycogen.}]\!] \\ = \; & causing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \end{aligned}$$

Both sentences have different semantics, yet we would like to return one as the match for the other one being the query. We can do it, because in the underlying ontology we have:

$$isa(forcing, causing)$$

Knowing that the action of "forcing" is kind of a "causing", the search engine shall in this case decide to return the sentence in question as a match.

We can conclude by saying that the incompleteness of the captured semantics can in fact improve the end result of the search.

5.9 Rephrasing

One sentence might be reformulated in different ways. Usually words can be moved around the sentence, in order to stress something, make the sentence funny, etc. The sentence normally keeps similar meaning.

Rephrasing, however, poses a challenge for the ontological text analyzer. Clearly, we would like to achieve semantics, which is independent of the way the sentence was phrased. This independence comes in most cases for free with the commutativity of the set intersection operation. As an example, consider:

$$\begin{aligned} & forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \\ = \quad & forcing \cap pnt : (storage \cap pnt : glycogen) \cap agt : insulin \end{aligned}$$

One could imagine that such two meanings are captured from different sentences. Nevertheless, they will be considered equal, as the set intersection ($\cap$) is commutative.

Another type of rephrasing is illustrated by the following example:

$$\begin{aligned} & [\![\text{Insulin forces storage of glycogen.}]\!] \\ = \quad & [\![\text{Insulin forces glycogen storage.}]\!] \\ = \quad & forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \end{aligned}$$

The system is able to construct the same semantics for this example, because it looks for patient of an action both in prepositional phrase and in compond noun phrase. This is the expected behaviour, as both sentences have the same meaning.

5.10 Relations vs. classes

For some sentences we have the choice of extracting semantics in two different ways: using a relation or a class. For our sample sentence we could have:

$$\begin{aligned} & [\![\text{Insulin forces storage of glycogen.}]\!] \\ = \quad & forcing \cap agt : insulin \cap pnt : (storage \cap pnt : glycogen) \end{aligned}$$

But we could also use the `CBY` relation, which relates an action to its cause. In such a case we would obtain:

$$\begin{aligned} & [\![\text{Insulin forces storage of glycogen.}]\!] \\ = \quad & storage \cap pnt : glycogen \cap cby : insulin \end{aligned}$$

Please observe how in one version the notion of forcing is represented by a class and in the other by relation.

It should be decided which of the two semantics is preferable. In our opinion the former is a better choice for the following reasons:

- By representing the action of forcing by the *cby* role we actually loose some useful information. This is because few other verbs could trigger this role, e.g. "causes".
- Since we did not introduce a subsumption relation between relations, but only between classes, representing forcing as a class allows for some more ontology–based reasoning. E.g. we can easily see that "forcing" is below "causing" in the ontology, which is in turn below "action", etc.
- One could argue that if we introduce a relation for representing the action of causing, we should be consistent and do the same for other actions as well. In our opinion it is preferable to keep the number of relations low.
- Relations also have the problem of requiring the inverse relations to be taken into consideration. Consider the following two semantics:

  $$storage \cap pnt : glycogen \cap cby : insulin$$
  $$insulin \cap cau : (storage \cap pnt : glycogen)$$

  Both semantics are equivalent, as the relation cau[4] is the inverse relation of cby[5]. For some reasoning engine, it must be explicitly defined that those relations are inverse, so that it could infer that the two presented semantics are the same.

5.11 Plurals

Sentences of our interest often involve plural constructions. Various construcions can be used for constructing plurals. In the following, we will concentrate on the "and" conjunction, which can form conjoined noun phrases, verb phrases, sentences, etc.

Let us consider the sentence:

A boy and a girl played. (5.6)

[4] $r(\phi, cau, \rho)$ means that phi causes ρ.
[5] $r(\phi, cby, \rho)$ means that phi is caused by ρ.

The sentence has two Logic of Plurals and Mass Terms readings – the collective one:

$$\exists x \exists y[boy'(x) \wedge girl'(y) \wedge played'(x \oplus y)]$$

and the distributive one:

$$\exists x[boy'(x) \wedge played'(x)] \wedge \exists y[girl'(y) \wedge played'(y)]$$

The distributive reading can be easily represented as two nodes in the generative ontology:

$$playing[agt : boy]$$

and

$$playing[agt : girl]$$

The collective reading is however problematic, as we need to decide how it should be captured. The three alternatives are discussed in the following sections.

5.11.1 Collectives operator

We could use the operator $\oplus$ for constructing plural sums of two individuals, as in:

$$playing[agt : (boy \oplus girl)]$$

The equivalent view in Peirce Algebra:

$$playing \cap \{X | \exists Y((X, Y) \in agt \ \wedge \ Y \in (boy \oplus girl))\}$$

In this proposal the $\oplus$ operator takes two sets, representing classes, and constructs a set representing collectives consisting of two individuals, each of which belongs to one of those two classes.

The advantage of this approach is that we can assume that relations have a functional character. It means that the intended meaning of the *agt* relation is as follows: We say that event e and individual i are *agt*–related ($(e, i) \in agt$) iff i participates in e as its only agent. If an event has more than one agent, we need to represent all of them as one collective agent, a plural.

5.11.2 Multiple semantic roles

We could simply use the same semantic role twice in a feature structure, as in:

$$playing[agt : boy, agt : girl]$$

The equivalent view in Peirce Algebra becomes:

$$playing \cap \{X | \exists Y((X,Y) \in agt \ \wedge \ Y \in boy)\} \cap \{X | \exists Y((X,Y) \in agt \ \wedge \ Y \in girl)\}$$

This means that the event which the sentence talks about is an intersection of three things:

- All events of playing;
- All events that have a boy as their agent;
- All events that have a girl as their agent.

We need to assume that relations do not have a functional character. It means that the intended meaning of the *agt* relation is as follows: We say that event e and individual i are *agt*–related ($(e,i) \in agt$) iff i participates in e as one of its agents.

5.11.3 Special nodes for representing collectives

We may introduce special classes in the ontology for representing plurals. Say, we have a class *sum*, representing a very general collection of objects. Then we can create the following node in the generative ontology as our representation of the sample sentence:

$$playing[agt : sum[cmp : boy, cmp : girl]]$$

We can think of this representation in the following way: the event belongs to the class *playing* and it has an agent, which belongs to a class *sum*, comprising two elements – a boy and a girl.

It is worth observing that this approach requires multiple semantic roles. The *sum* can be refined further into more specific classes, e.g. *combination*, *mixture*, etc.

5.12 Merging ontological semantics with categorial grammar

Let us now investigate how one could merge the ontological semantics developed in this chapter with categorial grammar.[6] The incorporation of ontologies can

[6]The combination of ontologies and categorial grammar has been presented previously by us in a paper at the International Multiconference on Computer Science and Information Technology – Computational Linguistics Applications.

be devised in order to reject ontologically–incorrect semantics to be assigned to some natural language fragments (e.g. "A table smiled."). Categorial Grammar is a well established tool for describing natural language semantics.[13, 34] Let us discuss some of its drawbacks and how it could be extended to overcome them. We use the extended version for deriving ontological semantics from text. A proof–of–concept implementation is also presented. We wish to extend the usual Categorial Grammar framework in order to achieve more flexibility. We also present how it can be used with an ontological component, which imposes well–formedness restrictions on sentences. The objective is to integrate formal ontologies with semantic domains.

Throughout this section we use classical Church's type theory, $\mathcal{C}$, as presented in [2]. We use the convention that functional types associate to the left, i.e. type $\gamma\beta\alpha$ is the same as $(\gamma\beta)\alpha$, which is also sometimes denoted as $\alpha \to (\beta \to \gamma)$.

5.12.1 Representing ontological semantics

We wish to construct ontological semantics for a fragment of English, following the approach in [24], using $\mathcal{C}$ as our formalism.

We can represent concepts appearing in a skeleton ontology as constants of type α: $Child_\alpha, Tall_\alpha, Running_\alpha, \ldots$. Those concepts are to be thought of as sets, e.g. $Child_\alpha$ is a set of all imaginable children. $Tall_\alpha$ is the set of all tall objects, thus properties such as "tall" are introduced in the ontology on the par with classes. $Running_\alpha$ is a set of all imaginable actions of running, etc. We do not want to be specific about α, but in the current example it could be replaced by $o\iota$, which is traditionally used for representing sets. Introducing nominalized verb forms as concepts in the ontology is in line with the adoption of the Davidsonian view. We also disregard meaning of determiners in the present fragment, as the resulting semantics is intended to be used for content–based text search. We will represent intersection of concepts using constant $\cap_{\alpha\alpha\alpha}$, e.g. $child \cap tall$ will be represented as $\cap_{\alpha\alpha\alpha} Child_\alpha Tall_\alpha$.

A simple skeleton ontology can be represented using a set of factual clauses of the form:

$$Sub_{o\alpha\alpha} Child_\alpha Person_\alpha$$
$$Sub_{o\alpha\alpha} Person_\alpha Physical_\alpha$$
$$Sub_{o\alpha\alpha} Person_\alpha Animate_\alpha$$
$$Sub_{o\alpha\alpha} Running_\alpha Action_\alpha$$

$Sub_{o\alpha\alpha}$ is a direct descendant relation, representing the lines that are drawn in the Hasse diagrams. $Isa_{o\alpha\alpha}$ can be specified as the (reflexive) transitive closure of $Sub_{o\alpha\alpha}$ in the following way:

$$\forall c_\alpha [Isa_{o\alpha\alpha} c_\alpha c_\alpha]$$
$$\forall c_\alpha \forall a_\alpha \forall p_\alpha [Isa_{o\alpha\alpha} c_\alpha a_\alpha \supset Sub_{o\alpha\alpha} c_\alpha p_\alpha \wedge Isa_{o\alpha\alpha} p_\alpha a_\alpha]$$

We use constants of type ρ to represent roles (binary relations), e.g. Agt_ρ, Loc_ρ. We can use Peirce product [12] to create compound concepts. We use constant $:_{\alpha\alpha\rho}$ for that purpose, e.g. $agt : child$ will be represented as $:_{\alpha\alpha\rho} Agt_\rho Child_\alpha$. Such a concept formation is well–known from Description Logics, where it would be represented as $\exists agt.child$.

For a sample sentence "The tall kid runs", we would like to derive the following ontologico–algebraic meaning:

$$running \cap agt : (child \cap tall)$$

This can be represented in type theory as a wff_α:

$$\cap_{\alpha\alpha\alpha} Running_\alpha [:_{\alpha\alpha\rho} Agt_\rho [\cap_{\alpha\alpha\alpha} Child_\alpha Tall_\alpha]]$$

We shall extend the specification of subsumption relation to accomodate for the intersection:

$$\forall x_\alpha \forall y_\alpha \forall z_\alpha [[Isa_{o\alpha\alpha} [\cap_{\alpha\alpha\alpha} x_\alpha y_\alpha] z_\alpha] \supset [Isa_{o\alpha\alpha} x_\alpha z_\alpha]]$$
$$\forall x_\alpha \forall y_\alpha \forall z_\alpha [[Isa_{o\alpha\alpha} [\cap_{\alpha\alpha\alpha} x_\alpha y_\alpha] z_\alpha] \supset [Isa_{o\alpha\alpha} y_\alpha z_\alpha]]$$

5.12.2 Modified Categorial Grammar

We introduce type ω. Constants of this type represent words in English, e.g. W_kid_ω, W_tall_ω, W_runs_ω. Notice that we use different names for constants denoting words and those representing concepts, e.g. W_tall_ω and $Tall_\alpha$. In this way they are not confused.

Let us define a new type, say, κ, which we will use for lexical entries. We also define three constants, which act as type constructors:

$$B_{\kappa(o\kappa\kappa)}$$
$$F_{\kappa(o\kappa\kappa)}$$
$$E_{\kappa\alpha}$$

Lexical entries are represented using the predicate constant $Lex_{o\kappa\omega}$. The lexicon consists of the set of factual clauses of the form:

$$Lex_{o\kappa\omega}W\_kid_{\omega}[E_{\kappa\alpha}Child_{\alpha}]$$

The type constructor $E_{\kappa\alpha}$ is used in the lexical entry in case the meaning of a word is some "fixed" ontological concept. The other constructors, $F_{\kappa(o\kappa\kappa)}$ and $B_{\kappa(o\kappa\kappa)}$ are reminiscent of the Categorial Grammar's backward and forward slashes, respectively. A lexical entry for word "tall" might look as follows:

$$Lex_{o\kappa\omega}W\_tall_{\omega}[F_{\kappa(o\kappa\kappa)}P\_tall_{o\kappa\kappa}]$$

In the above, $P\_tall_{o\kappa\kappa}$ is a predicate, which might be defined in the following way:

$$\begin{aligned}&\forall c_{\alpha}[[P\_tall_{o\kappa\kappa}[E_{\kappa\alpha}c_{\alpha}][E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}tall_{\alpha}c_{\alpha}]]]\\&\quad\supset[Isa_{o\alpha\alpha}c_{\alpha}Physical_{\alpha}]]\end{aligned}$$

The novelty in our approach is the use of predicates in meanings of words. In standard Categorial Grammar, lambda terms are used for that purpose. They are combined using β–reduction, with the only provision of categorial agreement. Consider the sentence "vitamin smiles." In traditional CG, "vitamin" has category np and meaning **vitamin**. The word "smiles" has category $np\backslash s$ and meaning $\lambda x.\mathbf{smile}(x)$. Since the categories fit together, the meanings get combined using β–reduction, and the sentence gets the meaning **smile**(**vitamin**). While the sentence is correct syntactically, it is incorrect from an ontological point of view. Unfortunately, usual CG does not allow us to introduce any ontological restrictions on the semantics. Our approach can reject this sentence, as we can use the following lexical entry for "smiles":

$$\begin{aligned}&Lex_{o\kappa\omega}W\_smiles_{\omega}[B_{\kappa(o\kappa\kappa)}P\_smiles_{o\kappa\kappa}]\\&\forall c_{\alpha}[[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}c_{\alpha}]\\&\qquad\qquad[E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha}[:_{\alpha\alpha\rho}\ Agt_{\rho}c_{\alpha}]]]]\\&\quad\supset[Isa_{o\alpha\alpha}c_{\alpha}Animate_{\alpha}]]\end{aligned}$$

The inability of β–reduction to fail has been already recognized as a problem by G. Ben–Avi and N. Francez in [9]. They have introduced a new formalism, which includes "β–reduction for ontologically–well typed λ–terms", among 12 definitions that constitute "the Ontological Lambek Calculus". Our proposal, however, has some further advantages:

- It is very formal – it is formalized fully within $\mathcal{C}$.
- It is very simple – it consists of a small number of formulae.
- It is very flexible – it allows adding arbitrary restrictions, e.g. one might like to use restrictions based on parthood relation rather than subsumption.
- It can be implemented in a straight–forward way, as presented in Section 5.12.4

For those reasons, rather than using β–reduction, we propose using a general proof system, which is a machinery having inherently the notion of failure at disposal. Now, desired semantic restriction can be handled by non–provability of a certain statement, e.g.:

$$K \nvdash \exists x_{\kappa}[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Vitamin_{\alpha}]x_{\kappa}]$$

In the above K denotes a set of formulas consisting of lexical assertions and rules introduced throughout this section. So a vitamin cannot smile, but a child certainly can:

$$K \vdash \exists x_{\kappa}[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Child_{\alpha}]x_{\kappa}]$$

Not only are we interested in the provability of such a goal, but we would also like to know what is the resulting semantics. In this case it is: $E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha}[:_{\alpha\alpha\rho} Agt_{\rho}Child_{\alpha}]]$

In other words:

$$\begin{aligned} K \vdash [P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Child_{\alpha}] \\ [E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha} \\ [:_{\alpha\alpha\rho} Agt_{\rho}Child_{\alpha}]]]] \end{aligned}$$

5.12.3 Elimination rules

Recall that the forward sequent rule of the natural–deduction Lambek Calculus takes the form:

$$\frac{a \Rightarrow \Phi_1 : A/B \qquad b \Rightarrow \Phi_2 : B}{a, b \Rightarrow \Phi_1(\Phi_2) : A}$$

and the backward rule:

$$\frac{a \Rightarrow \Phi_1 : B \qquad b \Rightarrow \Phi_2 : B\backslash A}{a, b \Rightarrow \Phi_2(\Phi_1) : A}$$

In the above a, b is the concatenation of a and b. Notice that the only provision for the elimination to take place is the agreement of syntactic categories. This is because $\Phi_1(\Phi_2)$ is always a well–formed λ–term, which can be β–reduced.

For the ease of reading, let us present the forward elimination rule of the generalized grammar in a similar, though quite informal way:

$$\frac{a_\theta \Rightarrow F_{\kappa(o\kappa\kappa)} p_{o\kappa\kappa} \qquad b_\theta \Rightarrow j_\kappa \qquad K \vdash p_{o\kappa\kappa} j_\kappa k_\kappa}{a_\theta, b_\theta \Rightarrow k_\kappa}$$

Let us formalize our generalized elimination rules in $\mathcal{C}$. The derivation relation ($\Rightarrow$) can be represented using a constant $R_{o\kappa\theta}$. The sequences of meanings will be represented using a well–known logic representation for lists. For the empty list we use the N_θ constant, and the list constructor is represented by $L_{\theta\theta\kappa}$.

The list concatenation can be represented using the list appending well–known from logic programming, though here defined for non–empty lists only:

$$\begin{aligned}\forall x_\kappa \forall y_\kappa \forall t_\theta [A_{o\theta\theta\theta} [L_{\theta\theta\kappa} x_\kappa N_\theta] \\ [L_{\theta\theta\kappa} y_\kappa t_\theta] \\ [L_{\theta\theta\kappa} x_\kappa [L_{\theta\theta\kappa} y_\kappa t_\theta]]]\end{aligned}$$

$$\begin{aligned}\forall h_\kappa \forall l_\theta \forall m_\theta \forall t_\theta [[A_{o\theta\theta\theta} [L_{\theta\theta\kappa} h_\kappa l_\theta] m_\theta [L_{\theta\theta\kappa} h_\kappa t_\theta]] \\ \supset [A_{o\theta\theta\theta} l_\theta m_\theta t_\theta]]\end{aligned}$$

The above use of $A_{o\theta\theta\theta}$ asserts that the concatenation of a and b yields g, or g is split into a and b. Now the forward elimination rule can be formalized in $\mathcal{C}$:

$$\begin{aligned}\forall a_\theta \forall b_\theta \forall g_\theta \forall j_\kappa \forall k_\kappa \forall p_{o\kappa\kappa} [R_{o\kappa\theta} g_\theta k_\kappa] \\ \supset [A_{o\theta\theta\theta} a_\theta b_\theta g_\theta] \\ \wedge [R_{o\kappa\theta} a_\theta [F_{\kappa(o\kappa\kappa)} p_{o\kappa\kappa}]] \\ \wedge [R_{o\kappa\theta} b_\theta j_\kappa] \\ \wedge [p_{o\kappa\kappa} j_\kappa k_\kappa]\end{aligned}$$

Similarly, the backward elimination rule:

$$\begin{aligned}\forall a_\theta \forall b_\theta \forall g_\theta \forall j_\kappa \forall k_\kappa \forall p_{o\kappa\kappa} [R_{o\kappa\theta} g_\theta k_\kappa] \\ \supset [A_{o\theta\theta\theta} a_\theta b_\theta g_\theta] \\ \wedge [R_{o\kappa\theta} a_\theta j_\kappa] \\ \wedge [R_{o\kappa\theta} b_\theta [B_{\kappa(o\kappa\kappa)} p_{o\kappa\kappa}]] \\ \wedge [p_{o\kappa\kappa} j_\kappa k_\kappa]\end{aligned}$$

We also need the following grammatical axiom, which expresses that if the list of meanings contains only one element, that element is the resulting meaning:

$$\forall k_\kappa [R_{o\kappa\theta} [L_{\theta\theta\kappa} k_\kappa N_\theta] k_\kappa]$$

The English text can also be represented formally in $\mathcal{C}$. We introduce a new type ζ for that purpose. We will represent the text as a list of words. An empty list of words will be represented by a constant $Tnil_\xi$ and a list constructor by a constant $T_{\xi\xi\omega}$.

5.12.4 Implementation outline

Using $\mathcal{C}$ instead of some kind of untyped logic as the underlying formalism has important advantages regarding the implementation. It forces us to think of the type of every formula, subformula and symbol. Thanks to that, the resulting formalization of the grammar is well–suited for implementation in a strongly–typed programming language. Using such a language (e.g. Mercury instead of Prolog) helps avoid very hard–to–find bugs and allows the compiler to generate much faster code.

We are interested in translating the formal specification given so far to a Prolog–like logic programming language, so that we can execute specific queries. For that purpose, we have formalized the grammar in $\mathcal{C}$ using only definite clauses. Furthermore, we perform the following steps:

- We write all the constants in lower–case
- We write all variables in upper–case
- We use `i` for intersection constant and `p` for the Peirce product constant
- We drop the subscripts indicating types in all formulas
- We replace '$\supset$' and '$\wedge$' with '`:-`' and '`,`', respectively.

- We add a period at the end of each formula.
- We replace curried notation of argument application with a non–curried one.
- We simply remove the universal quantification over all variables, as it is implicit.

Following all the given steps results in the program presented below. The result is a perfectly valid program in the Mercury logic programming language. Mercury is however strongly typed, so we need to add a few auxiliary definitions in order to compile it.

The type α can be modeled as follows:

```
:-type alpha ---> child; tall; physical;
  action; person; running;
  smiling; animate;
  ...
  i(alpha,alpha);
  p(role,alpha).
```

We need to add the following type, mode and determinism specification for the subsumption predicate:

```
:-pred sub(alpha::in,alpha::out) is nondet.
```

This tells Mercury that predicate `sub` takes an entity of type α as input, and computes an entity of type α as output. Furthermore, we specify that `sub` is a nondeterministic predicate, meaning that it can compute multiple outputs for one input. We need to provide similar specifications for all predicates in our program.

The factual subsumption database, translated directly from our previous definition in $\mathcal{C}$:

```
sub(child,person).
sub(person,physical).
sub(person,animate).
sub(running,action).
sub(smiling,action).
...
```

The type ρ of roles can be defined in Mercury as:

```
:-type role --->
    tmp ; loc ; prp ; wrt
  ; chr ; cum ; bmo ; cby
  ; cau ; cmp ; pof ; agt
  ; pnt ; src ; rst ; dst
  ; via ; ...
```

Clauses defining the `isa` relation:

```
isa(C,C).
isa(C,A):-sub(C,P),isa(P,A).
isa(i(X,_Y),Z):-isa(X,Z).
isa(i(_X,Y),Z):-isa(Y,Z).
```

The predicates included in lexical entries take the following form in Mercury:

```
p_tall(e(C),e(i(tall,C))):-
  isa(C,physical).

p_runs(e(C),e(i(running,p(agt,C)))):-
  isa(C,animate).

p_smiles(e(C),e(i(smiling,p(agt,C)))):-
  isa(C,animate).

p_the(e(I),e(I)).

...
```

The lexicon, translated to Mercury:

```
lex(w_kid,e(child)).
lex(w_tall,f(p_tall)).
lex(w_runs,b(p_runs)).
lex(w_smiles,b(p_smiles)).
lex(w_the,f(p_the)).
lex(w_a,f(p_the)).
...
```

The definition of non–empty list appending:

```
a(l(X,n),l(Y,T),l(X,l(Y,T))).
a(l(H,L),M,l(H,T)):-a(L,M,T).
```

The rules formalized in $\mathcal{C}$ take the following form in Mercury:

```
r(l(X,n),X).

r(G,E1):-
   a(G1,G2,G),
   r(G1,f(P)),
   r(G2,E2),
   P(E2,E1).

r(G,E1):-
   a(G1,G2,G),
   r(G1,E2),
   r(G2,b(P)),
   P(E2,E1).
```

The rules are the only place in the program, where we use higher–order predicates.

The following predicate assigns a meaning to English text:

```
m(T,C):-
        map_lex(T,L),
        r(L,e(C)).
```

Let us add an auxiliary predicate `work` for testing purposes:

```
work(C):-
  m(t(w_the,
      t(w_tall,
        t(w_kid,
          t(w_smiles,
            tnil)))),C).
```

The `map_lex` predicate is used for lexicon look–up of a list of words.

```
map_lex(tnil,n).
map_lex(t(W,T),l(I,MT)):-
   lex(W,I),map_lex(T,MT).
```

Notice that the program does not require higher order unification, except for the simplest case where a variable is bound to a predefined predicate and such a predicate is called. Most logic programming languages provide a facility for such behaviour. We could change the syntax slightly and turn our code into a valid Prolog program by utilizing the `call` library predicate.

The full listing of the program is given in Appendix C on page 207.

Notice that the program is a direct implementation of the theory, hence quite inefficient. Instead of the presented top–down approach, we could derive the semantics in the bottom–up manner in order to avoid unneeded search.

For the sample query:

```
? work(C).
```

We get the following (exactly one) result:

```
C = i(smiling, p(agt, i(tall, child)))
```

5.13 Summary

In this chapter we have introduced the notion of ontological semantics. In particular we will focus on the Peirce–algebraic ontological semantics, as it fits perfectly with the generative ontology.

Furthermore, we have presented an example of how our ontological semantics can be utilized by merging it with Categorial Grammar. The functional composition and β–reduction have been replaced with proof rule application. We have utilized it for constructing ontologico–algebraic meaning using ontological restrictions, dropping at the same time syntactic categories and syntactic restrictions.

Chapter 6

Comparison with state–of–the–art in ontological semantics

In this brief chapter, as an interlude before we commence with developing systems for ontological meaning extraction, let us have a look at the state–of–the–art and compare it to our approach.

One of the state–of–the–art approaches to ontological semantics utilizing Lambek Calculus has been presented by G. Ben–Avi and N. Francez in [9]. We have previously presented a discussion of this work and suggested an alternative approach in Section 5.12 on page 98.

6.1 Comparison to "Ontological Semantics" by Nirenburg and Raskin

Perhaps it is worth comparing our approach to the ontological semantics described in the previous chapter to that described in the book by the same title by Nirenburg and Raskin [33].

- In our approach we use First Order Predicate Logic as the underlying formal metalanguage in order to maintain coherence between theory, specification, and implementation parts. Nirenburg and Raskin (N&R) dismiss

First Order Predicate Logic, as according to them it has failed "to have made a historical impact". We try to remain formal in our discussions, while they tend to be more philosophical and informal.

- N&R deal with textual natural language semantics in general. We concern ourselves only in general with scientific texts, particularly biomedical ones. This has large impact on the way ontologies are used in the semantics. Due to unrestricted domain, N&R' semantics approaches issues of aspect, detailed time, language style (formality, politeness, respect, force, simplicity, color, directness), modality, etc. These issues are not relevant for our scientific text domains.
- The tasks that N&R's text meaning representations have been used for include machine translation, information extraction, question answering, general human–computer dialog systems, and text summarization. This is a wide range of linguistic applications, while we focus ourselves on the problem of semantic search.
- Our approach is unique in that we use the notion of generative ontologies in order to establish a formal ontological semantics.
- We provide several concrete algorithms for extracting ontological semantics from text (and one for generating text given ontological semantics), with implementation. While N&R discuss how text meaning representations can be extracted from text, they do not provide a specific algorithm or implementation thereof.

6.2 Framenet

Let us consider the Framenet project [6, 49, 7], centered around the idea of semantic frames.

In [15] we read:

> The basic idea is that one cannot understand the meaning of a single word without access to all the essential knowledge that relates to that word. For example, one would not be able to understand the word "sell" without knowing anything about the situation of commercial transfer, which also involves, among other things, a seller, a buyer, goods, money, the relation between the money and the goods, the relations between the seller and the goods and the money, the relation between the buyer and the goods and the money and so on.

This approach differs substantially from ours in various respects. As it appears, frame semantics presuposes the need of quite large amount of real–world knowledge. The quoted "knowing about the situation of commercial transfer" is very advanced and complicated task with a computer. Full formalization of such a knowledge is an extremely difficult task – so difficult in fact, that nobody has ever come close to accomplishing it (for non–abstract concepts). All existing real–world knowledge formalization attempts simplify the view of the problem, and describe only some relatively small (but usually most important) part of it, leaving the less–important (or sometimes just more–difficult–to–model) elements out. Cf. e.g. CYC [27]. FrameNet serves as a simplified view of this "knowing", where frames are used to encode some limited information about situations like "commercial transfer". However, comparing FrameNet to our approach, it still contains much more information in the frames than we do in our setup. This has both advantages and disadvantages. The advantage of using FrameNet for our purposes could be that the information available could improve the understanding of language. That's the goal of FrameNet or any other lexical resource – more knowledge allows better language comprehension. However, the downside is that one requires a large amount of frames. At the time of writing FrameNet consists of close to 800 frames, which covers an impresive amount of language. Nevertheless, still a lot more is needed to claim a very large linguistic coverage. Projects like BioFrameNet [16] try to fill the gaps for some domains, but as with any knowlegde representation project, a lot is left to do.

In a sense, in our approach, we would like to have quite large language coverage. Therefore we only depend on a lexicon and a skeleton ontology, both of which can support very large linguistic coverage, far beyond FrameNet's 800 frames.

The key difference, however, lies hidden in the underlying motivation and aspirations of the projects. Once again, frame semantics claims:

> The basic idea is that one cannot understand the meaning of a single word without access to all the essential knowledge that relates to that word.

While we fully agree with this statement, the point to notice is that, as far as our approach is concerned, we refrain from trying to "understand the meaning of a single word". Instead of understanding the full meaning of a word, a phrase, or a sentence, we limit ourselves to capturing what the sentence is about, i.e. extracting conceptual information from the sentence and linking to the generative ontology. This comes from the fact that our ontological semantics

is supposed to be used for search, and for that purpose we do not need to understand the propositional content of a sentence. Hence, in our approach, understanding a word is limited to linking the word to a concept in the generative ontology.

Also, according to [39], FrameNet cannot be used as an ontology.

Even though for the reasons explained above, we do not use FrameNet directly, some of its data could still be utilized while building a generative ontology. FrameNet contains semantic categories for frame elements, and those could be manually[1] turned into ontological affinities that define the generative ontology. However, we do not pursue this idea.

[1] It seems unlikely to achieve good results with automatic methods.

Chapter 7

The computational view of the relation between the natural language fragment and the ontological semantics

This chapter gives an overview of how one could relate a fragment of the natural language and the ontological semantics established in chapter 5 by computational means. We call the process of extracting the ontological semantics from text "ontograbbing". However, we also discuss the reverse problem of linguistic realization, i.e. transforming a given ontological semantics into a linguistic form, such as a phrase or a sentence.

The chapter starts with a simple definite clause example of ontograbbing, which algorithmically relates text to its ontological semantics. This relation is supposed to be very formal, so the language is limited and narrowed down to selected language fragments. Further extensions are presented, in order of increasing complexity. In the next chapter we will present a more robust model accepting realistic text.

The algorithms presented in this chapter are not meant to be efficient, but to reflect closely the logical relation between language and its semantics established

in the previous chapter.

Since these simple ontograbbers are implemented in Prolog (including its special Definite Clause Grammar notation), a short discussion about suitability of that system is provided.

7.1 Computation with Prolog vs. specification in First Order Predicate Logic

In case of First Order Predicate Logic, we could use a Prolog system as a convenient computational approximation that would allow us to reason in an efficient way. However, Prolog systems use most commonly some version of SLD resolution coupled with closed world assumption and negation as failure in order to simulate the First Order Predicate Logic reasoning. The word "simulate" is used intentionally here. Let us have a look at how First Order Predicate Logic and Prolog logic programming differ. In the latter, due to the fact that SLD resolution is used, we are limited to writing our sentences as definite clauses.

This poses serious restrictions on our freedom to use arbitrary operators and quantifiers anywhere within our sentences. We are limited to using definite clauses only, which are all built as universaly quantified implications with one term on the left hand side and a conjunction of arbitrary number of terms on the right hand side. Unfortunately, not every First Order Predicate Logic formula can be translated to a set of definite clauses to be used by Prolog systems. One of the most noticeable problems with such translation is the fact that Prolog uses negation as non–provability instead of classical negation.

Prolog systems often offer additional facilities, e.g. cut operation, which go beyond definite clauses. Such extensions modify the way SLD resolution operates, and hence logic programs containing them are referred to as impure.

However, even if the set of formulae is pure and exactly the same for First Order Predicate Logic and Prolog, it is still not to be understood in the same way. We have to keep in mind how Prolog executes the given logic program. The order of clauses is very important. Swapping the order of two clauses can turn a well–working program into a non–terminating one. We do not observe such shortcomings with First Order Predicate Logic. Similar issue occurs with the order of literals in the body of a given clause. While the order of literals in a conjunction is strictly arbitrary in First Order Predicate Logic, it matters a lot with Prolog.

This situation means that while some First Order Predicate Logic specifi-

cation can provide a full specification of some problem, a corresponding logic program may only reflect a single side of it. This means that some Prolog queries, which are sentences to be proven, might work well with the given set of clauses, while others may cause Prolog to loop infinitely.

For the reasons described above, a given logical specification may have to be rewritten as several logic programs, each allowing to pose some kind of queries. This situation is often encountered if we want Prolog to perform synthesis and analysis at the same time. Often one is achievable, while the other causes queries to loop indefinitely, or vice versa. The issue may be fixed by writing one logic program to be used with synthesis queries and another to be used with analysis queries. Such a solution may be regarded as not fully satisfactory, as both programs usually differ only in the order of clauses or literals within clause bodies.

7.2 Lists

In the tradition of logic programming, lists are used for representing sequences of tokens or words. Most logic programming languages use a special notation for lists, where a instead of writing $list(head, tail)$ one writes `[head|tail]`. Also instead of using the recursive notation `[a|[b|[c|...]]]` one uses `[a,b,c,...]`.

In all presented ontograbbers we make extensive use of lists. We use them for representing all kind of phrases and sentences, as lists of words. Furthermore, they will also play key role in representing semantic structures, what is explained further in Section 7.5.1 on page 127.

7.3 Computing with Definite Clause Grammars

Definite Clause Grammars are simply grammars expressed as definite clauses [35]. They allow us to turn the problem of language parsing or generation into theorem proving task. The grammar for the language needs to be expressed in first order logic in form of definite clauses. The fact that the clauses are definite facilitates the use of resolution systems for finding the proofs. The most common such a system is Prolog, which uses SLD-resolution with unification for this task.

Hence, using DCGs not only allows us to provide formal ascriptions of ontological semantics to text, but also function as a practical computational method. However, the practicality of this method is questioned by some limitations of the logic programming paradigm itself. Some of those limitations stem from

the discrepancies between the underlying logic with its proof system and the inferential engine used to simulate it in logic programming in a computational manner.

7.4 Ontograbbing with definite clauses

We first present a simple grammar, which does not make use of the special DCG notation available in Prolog. All definite clauses are "spelled out" fully. Additionally, for simplicity, we do not use difference lists, which would be hidden behind the special Prolog DCG notation.

In the course of analyzing the sentence the list of words needs to be divided into shorter sublists, corresponding to shorter phrases or words. For the purpose of (nondeterministically) dividing any list of tokens into two shorter lists we use the `split` predicate:

```
split(X,[H1|T1],[H2|T2])
  :-append([H1|T1],[H2|T2],X).
```

The approach to parsing employed for this simplest ontograbber is by no means the fastest one. However, it allow us to provide a clean logical description of how the ontological semantics is coupled with the grammar. Describing an efficient parsing process, e.g. utilizing the Earley[1] parser would call for a very large number of formulae in First Order Predicate Logic.

7.4.1 Noun phrases

The `np` predicate defines the logical relation between noun phrases and corresponding ontological concepts. The singleton list containing only one token `N` can be recognized as a noun phrase corresponding to the concept `C` if the lexicon contains an entry for the noun `N` with an associated meaning `C`:

```
np([N],C):-noun(N,C).
```

For handling noun–noun compounds, we have the following rule:

```
np(L,C):-split(L,L1,L2),np(L1,C1),np(L2,C2),
         a_nnc(C1,C2,C).
```

[1] An SML implementation of the Earley parser is presented in Appendix D.

Here we assign onto–meaning `C` to a noun–noun compound `L` provided that it splits into two shorter noun phrases `L1` and `L2` with onto–semantics `C1` and `C2`. An additional requirement is that the ontological affinities allow us to combine the meanings `C1` and `C2` into the meaning `C`, which is expressed in terms of the `a_nnc` predicate.

7.4.2 Handling prepositional phrases

First, let us observe that we do not handle prepositional phrases (PP) themselves. We always analyze them together with the phrase to which they are attached. Hence, with noun phrases (NP) we will look at the tripples of the form NP_1 P NP_2 instead of NP_1 PP, where PP would consist of P and NP_2. The reason for this is that we use all three elements for ontological disambiguation of the role carried by the preposition. This is discussed in detail in Section 7.4.6 on page 120.

```
np(L,C):-split(L,L1,L2),split(L2,[P],L3),
         prep(P),np(L1,C1),np(L3,C3),
3        a_pp(C1,P,C3,C).
```

This rule splits the (candidate) noun–phrase `L` into three sublists by performing the splitting action twice. First `L` is split into `L1` and `L2`, and then `L2` is split into a singleteon list containing only one preposition `P` and remaining list `L3`.

After the splitting, we perform three actions. We test whether `P` is actually a preposition according to our lexicon and we recursively apply noun phrase recognition rules for both `L1` and `L3`. Notice that we could have performed those actions also in more "natural" order, that is the order of the elements in the phrase `L`. The resulting rule would be as follows:

```
np(L,C):-split(L,L1,L2),split(L2,[P],L3),
2        np(L1,C1),prep(P),np(L3,C3),
         a_pp(C1,P,C3,C).
```

Nevertheless, we choose not to do it, as the "unnatural" order has higher efficiency. This is due to the fact that testing whether `P` is actually a preposition takes a very short time, as it only requires one lookup in the lexicon. But testing whether `L1` is a noun phrase (and calculating its ontological semantics) may take a longer time. If `L1` would not turn out to be a correct noun phrase, the system

would need to backtrack only to find perhaps that in fact `P` is not a preposition. Hence, testing the preposition first improves the performance of the system by avoiding a lot of unnecessary backtracking. After all, if we encounter a preposition in a sentence, a fairly good guess is that it follows a noun phrase.

Such a shuffling of atoms in the body of the clause is specific to practical logic programming systems. We would never consider these two rules to be any different if we would be working with logical specification in First Order Predicate Logic.

The question arises – what if there is more than one prepositional phrase attached to some noun phrase? In such a situation we observe ambiguity at the syntactic level, i.e. the prepositional phrases can be either alligned or nested. However, it turns out that our rule for handling noun–preposition–noun triplets is flexible enough to handle these cases. This stems from the fact that the rule is recursive in nature. Let us investigate how the rule operates in such cases by considering the following example:

$$\underbrace{\text{Blood flow through the liver after meals}}_{NP0}$$

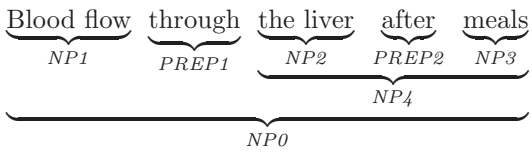
The rule can split the phrase into three parts in two ways (as there are two prepositions):

$$\underbrace{\underbrace{\text{Blood flow}}_{NP1}\ \underbrace{\text{through}}_{PREP1}\ \underbrace{\text{the liver after meals}}_{NP4}}_{NP0}$$

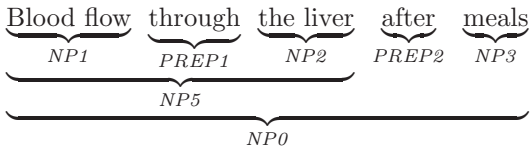
or

$$\underbrace{\underbrace{\text{Blood flow through the liver}}_{NP5}\ \underbrace{\text{after}}_{PREP2}\ \underbrace{\text{meals}}_{NP3}}_{NP0}$$

In both cases, after the splitting is performed, the rule is called recursively on the sublists, so that the noun phrases containing prepositions (*NP4* and *NP5*) will be handled using the same rule, in effect splitting them again into three parts each. So, after the second phase of splitting, we will be left with the following two divisions:



or



As we can observe, in both cases the final constituents to contribute to the ontological semantics are the same, as could be expected. However, they are structured differenctly at a higher level, via either *NP4* or *NP5* corresponding to either nested or alligned prepositional phrases, respectively.

7.4.3 Determiners

The following rule handles determiners. Notice that determiners do not contribute to the ascribed meaning:

```
np([D|L],C):-det(D),np(L,C).
```

Any determiners in the analyzed sentence are simply discarded. This has an interesting side–effect. If the sentence contained determiners which are not soundly placed according to the grammar, the analysis will continue without noticing it. Usually a sentence containing e.g. a list of tokens “the the the” should be rejected, as being ungrammatical.

7.4.4 Adjectives

Adjectives can be analyzed in the following way, analogous to how we dealt with other phrases:

```
np(L,C):-split(L,[A],L1),adj(A,CA),np(L1,C1),
        a_adj(CA,C1,C).
```

7.4.5 Sentences

Sentences built of a subject NP, a transitive verb, and an object NP, will be analyzed as follows:

```
s(L,C):-split(L,L1,L2),split(L2,[V],L3),
        np(L1,C1),tv(V,CV),np(L3,C3),
        a_tv(C1,CV,C3,C).
```

7.4.6 Ontology–based role recognition

Noun–noun compounds

An interesting problem is the recognition of a proper relation between concepts in a noun–noun compound. While in prepositional phrases the preposition contains a hint as to what role should be ascribed in the resulting semantics, noun–noun compounds are more tricky. This recognition, in the current prototype, is made on the basis of where the concepts corresponding to the constituent nouns are placed in the ontology. E.g. if the meaning of the modifier noun is below "substance" in the ontology, and the meaning of the head noun is below "process", we recognize the role to be "pnt":

```
a_nnc(X,Y,i(Y,p(pnt,X))):-isa(X,substance),
                          isa(Y,process).
```

Examples of this situation are: "glucose storage", "insulin secretion".
Similarly, we can base the role recognition on the parthood relation:

```
a_nnc(X,Y,i(Y,p(loc,X))):-partof(X,body),
                          partof(Y,body).
```

If both nouns represent parts of the body, we understand the hidden relation between them to be "loc". E.g. "liver cell".

Prepositional phrases

When used to represent compound concepts, prepositional phrases give us more information about the intended role between the partial concepts than noun–noun compounds do. The clue is the preposition used. Nevertheless, prepositions still do not allow us to determine the role uniquely, as they are in general

ambiguous. We can, however, similarly as with noun–noun compounds, use the ontology for the purpose of disambiguation. In the current theory, the predicate used for role disambiguation in prepositional phrases is called `a_pp`.

It is worth noting that the disambiguation need not be absolute, i.e. we will not always get one unique role. Out of n possible roles, we will get m disambiguated ones, where $0 \leq m \leq n$. Thus, we will usually get fewer roles, most commonly one or two. It can also happen that we do not get any role at all in the disambiguation process. This happens if our ontological affinities are unable to "understand" a given prepositional prase type from ontological perspective. Such a situation is common in domain–specific texts, if the given text falls outside of the domain considered, but the ontology is only defined for the domain.

The `a_pp` predicate might be defined as follows:

```
a_pp(X,of,Y,i(X,p(pnt,Y)))
  :- isa(Y,substance),
3     isa(X,process).
```

Such a rule defines that if the noun phrase corresponding to concept `X` is augmented with a prepositional phrase consisting of the preposition "of" and a nounphrase corresponding to the concept `Y`, then the resulting role can be "pnt", provided that `X` is below "process" in the ontology and `Y` is below "substance". The resulting compound concept will be `i(X,p(pnt,Y))`, which can be written in the feature–structure notation as $X[pnt : Y]$.

As an example, let us consider the prepositional phrase "secretion of insulin". According to the disambiguation rule above, we can understand the preposition "of" as representing the role *pnt*, due to the fact that "secretion" falls below "process" and "insulin" falls below "substance" in our ontology.

Note that it would be difficult to disambiguate the role represented by the prepositional phrase by looking at the phrase itself. That is why we also investigate the noun phrase to which this prepositional phrase is attached. As an example of why this is the case, consider the phrase "of insulin". By looking at such a prepositional phrase itself, it is difficult to decide what role the preposition denotes. If this phrase is attached to the noun "secretion", we would understand it as *pnt*. However, if the noun would be "level", thus forming the phrase "level of insulin", then we would want to understand the role to be *wrt*.

Obviously, our role disambiguation for prepositional phrases can use arbitrary definitions for any given preposition. We are not limited merely to taxonomical inclusions. We could, for instance, also use parthood information for

this purpose, as shown in the following example:

```
a_pp(X, across, Y, i(X, p(via, Y)))
  :- isa(X, transport),
     partof(Y, body).
```

In this case, if the noun phrase corresponding to concept `X` is augmented with a prepositional phrase consisting of the preposition "across" and a nounphrase corresponding to the concept `Y`, then the resulting role can be "via", provided that `X` represents some kind of transport process in the ontology and `Y` is some body part. The resulting compound concept will be `i(X,p(via,Y))`, or in the familiar feature–structure notation: $X[via : Y]$.

Transitive verbs

We will use similar principles for understanding roles intended to hold between concepts expressed as a transitive verb and the corresponding subject and object. Similarly as with noun–noun compounds, and contrary to the prepositional phrases, the roles are not stated in any way. The most important hint for role understanding is that transitive verbs tend to be used most commonly when expressing agent and patient roles. Other linguistic constructs are rarely used for this purpose if subject–transitive-verb–object can be used. Of course the order is a big clue in itself, as the agent role holds between the verb's concept and the subject's concept, while the patient role ties the verb's concept with object's concept. This is the case with verbs in active form, and those are under current consideration.

We will use the `a_tv` predicate for defining the relation between the concepts corresponding to the verb, the subject, the object, and the resulting ontosemantics. As an example, consider the following rule:

```
a_tv(X,Y,Z, i(Y, i(p(agt,X), p(pnt,Z))))
  :- isa(Y, influence).
```

This rule states quite simply that for any verb, which represents some concept that can be found under "influence" in the ontology, we can analyze the relation between the verb's concept and subject's concept as "agt" and between verb's concept and subject's concept as "pnt". Verbs that fall into this group include "to inhibit", "to promote", etc. Observe that the only condition in the body of the definite clause is taxonomical inclusion regarding the verb's concept. We do not pose any requirements on the subject's and object's concepts. This

means that in a sample sentence "A inhibits B", no matter what A and B are, we will assume the outcoming roles to be "agt" and "pnt".

Adjectives

The `a_adj` predicate is used for defining the relation between the adjective's concept, noun's concept, and whole phrase's concept. The constraints for this relation to hold are ontologically–based, as in the following example:

```
a_adj(X,Y,i(X,Y)):-isa(X,position_on_scale),
                   isa(Y,level).
```

Here if the concept represented by the adjective is some kind of "position_on_scale", and the noun represents something below "level", we understand the concept of the whole phrase to be the intersection of these two concepts. This follows the view that adjectives can be understood as classes rather than properties, e.g. that everything that is green constitutes the class of green things.

The desired ontosemantics may also express some adjectives in terms of Peirce product including some desired role. For instance, we may like to use the adjective "pancreatic" as meaning "loc:pancreas". The `a_adj` predicate is able to handle such a situation easily. Here is an example of how this can be achieved:

```
a_adj(X,Y,i(p(chr,X),Y))
  :-isa(X,position_on_scale),
3 isa(Y,level).
```

One of the advantages of this approach is that we can define ontosemantics for some groups of adjectives to follow the intersective approach, and for other groups to follow the Peirce product approach.

7.4.7 Expressing ontology using definite clauses

Definite clauses are not only well suited for expressing the grammar. They can be easily employed for representing the ontology.

The direct taxonomical subsumption can be formulated as a collection of facts:

```
isa(inhibition,influence).
2 isa(promotion,influence).
```

```
isa(insulin,substance).
isa(glucose,substance).
isa(transport,process).
isa(low,position_on_scale).
isa(high,position_on_scale).
...
```

We can as easily represent the parthood relation database:

```
partof(cell,body).
partof(membrane,body).
...
```

7.4.8 Ambiguities

Prolog is quite unique in the easiness of expressing ambiguities. Many other programming languages lack that property, e.g. imperative or functional languages. In those one has to explicitly implement some handling of ambiguities – say, a function would need to return a set of answers instead of one answer, and of course a lot of changes in the code are necessary to handle that.

Fortunately, with Prolog, ambiguities are handled very easily due to the mechanism of SLD resolution. We can include as many definite clause as we need, and the system will try them all, possibly producing many answers instead of just one. Such a behavior is perfect for expressing, e.g. the ambiguities for lexical items or ontological bindings.

7.4.9 Expressing the lexicon using definite clauses

Similarly as with the ontology, we can represent the lexicon in a straightforward manner. An interesting question arises – Prolog uses the closed world assumption – is this adequate for lexical representation of both closed and open parts of speech? The answer is yes. Although we might be mislead by the similarity of naming, the open world assumption, which First Order Predicate Logic uses, is not very useful for representing the open classes of words.

Suppose we would represent our lexicon in a formalism which utilizes the open world assumption. Then for any given word we would need to specify not only which part(s) of speech it is, but also list a lot of negative information

about which parts of speech it is not. This would rather be an impractical approach.

The confusion arises from the fact that the word "open" in "open world assumption" refers to the openness of the knowledge about a given word – we do not know that it does not belong to some class unless we specify that. But the word "open" in "open word class" refers to the openness of the class as a whole, meaning that it does not have a fairly fixed amount of member words.

Determiners

The determiners are defined by the `det` predicate:

```
det(these).
det(the).
...
```

Prepositions

We take care of prepositions just as easily:

```
prep(of).
prep(across).
...
```

Nouns

The predicate for noun lexical entries will be binary, coupling the given noun with its ontological class:

```
noun(transport,transport).
noun(insulin,insulin).
...
```

Verbs

Transitive verbs are stored in the lexicon together with the ontological class they represent by means of the `tv` predicate:

```
tv(inhibit,inhibition).
...
```

Adjectives

Lexical entries for adjectives are taken care of in the same way as nouns and verbs:

```
adj(low,low).
...
```

7.4.10 Example

Now, using Prolog's SLD resolution, we can for instance ask the following query:

```
:-s([these,low,insulin,level,inhibit,
     the,transport,of,glucose,across,cell,
         membrane],
    C)
```

We will get only one answer:

```
C = i(inhibition,
      i(p(agt, i(low, i(level,
                        p(wrt, insulin)))),
        p(pnt, i( i(transport,
                    p(pnt, glucose)),
                  p(via, i(membrane,
                           p(loc, cell)))))))
```

7.5 An extended version of the ontograbber

We now present another ontograbber expressed with definite clauses to be used with the Prolog logic programming language.

This version uses similar basic principles as the previous one, except it is extended in many respects. Let us go step by step through the changes.

7.5.1 Noun phrases

In the current prototype we employ an alternative representation for ontoterms, where we make use of lists for representing intersections of classes. This is similar to the conjunctive normal form representation of formulas. In our new notation the intersection

$$a \cap b \cap c$$

is represented as a list

```
[a,b,c].
```

Hence, we can compute intersections of ontoterms simply by appending two lists and sorting the result:

```
i(C1,C2,C):-append(C1,C2,C3),sort(C3,C).
```

This solves the problem of representing intersections using the `i(A,B)` function symbol. Such a representation is not handled easily due to the fact that two expressions

$$(a \cap b) \cap c$$

and

$$a \cap (b \cap c)$$

are represented by two different terms. In the new representations both expressions will take form of the same list.

For that reason we need to refine our grammatical definition of noun phrases, which includes the lexicon lookup:

```
np([N],[SC]):-noun(N,SC).
```

We introduce the distributive reading of conjunctions at noun phrase level using the following rules:

```
np(L,C):-split(L, L1,[and|_L2]),np(L1,C).
np(L,C):-split(L,_L1,[and| L2]),np(L2,C).
```

Another small change concerning noun phrases is the tabularisation of definitions for role disanbiguation in noun–noun compounds. The information necessary for disambiguation will be located in a "table" called `t_nnc`. This table is looked up by our disambiguating predicate `a_nnc`:

```
a_nnc(X,Y,Z):-t_nnc(A,B,R),isa(X,A),isa(Y,B),
                    i(Y,[p(R,X)],Z).

t_nnc(chemical_compound,process,pnt).
t_nnc(substance,level,wrt).
t_nnc(physical_object,physical_object,loc).
...
```

7.5.2 Prepositional phrases

Handling of the prepositional phrases is modified, as we would like to add the recognition of conjunctions. We face a problem – how do we take care of the recognition of the intended reading of the confunction, i.e. whether it is distributive or collective?

According to the study we have conducted (see Appendix A on page 181) the reading of "and" is almost exclusively distributive in biomedical scientific texts. The collective reading is sometimes present at the level of prepositional phrase and is always hinted by the use of some noun representing collectivity, e.g. "mixture of A and B", "combination of A and B", etc. This requires some kind of "exception" mechanism to be utilized, as we would like the reading to be distributive by default, and collective for some special cases. We can model this situation by using the so–called negation as failure, which is an impure facility offerred by Prolog systems. Strictly using negation as failure results in the program which goes beyond definite clauses. Any clause containing this impurity cannot be regarded as a definite clause.

Hence, we will have more rules for handling prepositional phrases. We use the ontology for the purpose of detecting whether the reading is distributive or collective. We inspect the concept corresponding to the noun to which the prepositional phrase is attached. If that noun represents some kind of "combination", then we assume the reading to be collective:

```
np(L,C):-split(L,L1,[P|L2]),
         prep(P),np(L1,C1),
         isa(C1,[combination]),
         split(L2,L21,[and|L22]),
         np(L21,C21),np(L22,C22),
         a_pp(C1,P,C21,C01),
```

```
        a_pp(C1,P,C22,C02),
        i(C01,C02,C).
```

Otherwise, the reading is understood to be distributive:

```
np(L,C):-split(L,L1,[P|L2]),
        prep(P),np(L1,C1),
        \+isa(C1,[combination]),
        np(L2,C2),
        a_pp(C1,P,C2,C).
```

Another extension in this version of the ontograbber is the tabularization of the ontological role disambiguation information.

We will store the many–to–many relation between prepositions and roles in the `p_r` predicate:

```
p_r(of,pnt).
p_r(by,agt).
p_r(across,via).
p_r(of,loc).
p_r(of,wrt).
p_r(through,bmo).
p_r(of,cmp).
...
```

The information about possible roles between some top–level concepts has been extracted into the `t_pp` table:

```
a_pp(X,P,Y,Z):-t_pp(A,R,B),p_r(P,R),isa(X,A),isa
    (Y,B),i(X,[p(R,Y)],Z).

t_pp(process,pnt,substance).
t_pp(influence,pnt,process).
t_pp(process,agt,univ).
t_pp(transport,via,univ).
t_pp(body_part,loc,person).
t_pp(property,wrt,univ).
t_pp(influence,bmo,process).
t_pp(combination,cmp,univ).
...
```

7.5.3 Genitives

In this extension of the ontograbber we introduce handling of genitives:

```
np(L,C):-split(L,L1,[s|L2]),
         np(L1,C1),
         np(L2,C2),
         a_gen(C1,C2,C).
```

The role recognition for genitives is taken care of in the same way as for noun–noun compounds:

```
a_gen(X,Y,Z):-t_gen(A,B,R),
              isa(X,A),
              isa(Y,B),
              i(Y,[p(R,X)],Z).

t_gen(person,body_part,pof).
t_gen(univ,process,agt).
...
```

7.5.4 Sentence level grammar

At the level of sentences, conjunctions have only distributive meaning and the following rules model that:

```
s(L,C):-split(L,  LS1,[and|_LS2]),s(LS1,C).
s(L,C):-split(L,_LS1,[and|  LS2]),s(LS2,C).
```

7.5.5 Verb phrases

Again we introduce two new rules to cope with distributive conjunctions, this time at verb phrase level:

```
s(L,C):-split(L,LNP1,[VPH|VPT]),tv(VPH,_),
        split([VPH|VPT],LVP1,[and|_LVP2]),
```

```
        append(LNP1,LVP1,L1),
        s(L1,C).

s(L,C):-split(L,LNP1,[VPH|VPT]),tv(VPH,_),
        split([VPH|VPT],_LVP1,[and|LVP2]),
        append(LNP1,LVP2,L2),
        s(L2,C).
```

We analyze verb phrases without conjunctions at the sentence level in the same way as before:

```
s(L,C):-split(L,LNP1,[TV|LNP2]),tv(TV,TVC),
        np(LNP1,NPC1),np(LNP2,NPC2),
        a_tv(NPC1,TVC,NPC2,C).
```

However, we introduce another extension, i.e. the possibility of attaching prepositional phrases at the verb phrase level through the following rules:

```
s(L,C):-split(L,LNP1,[TV|L2]),
        tv(TV,TVC),
        split(L2,LNP2,[LPPSH|LPPST]),
        prep(LPPSH),
        np(LNP1,NPC1),
        np(LNP2,NPC2),
        pps(TVC,[LPPSH|LPPST],TVC1),
        a_tv(NPC1,TVC1,NPC2,C).

pps(C,[],C).
pps(C,LPPS,C2):-
        append([P|NPL],LPPS1,LPPS),
        prep(P),
        pps(C,LPPS1,C1),np(NPL,NPC),
        a_pp(C1,P,NPC,C2).
```

We also tabularize the ontological information for role disambiguation and make use of the more efficient representation for concept intersections:

```
a_tv(X,Y,Z,V):-t_tv(A,RS,RO),
                isa(Y,A),
                i(Y,[p(RS,X)],W),
```

```
                    i(W,[p(RO,Z)],V).

t_tv(influence,agt,pnt).
t_tv(process,agt,pnt).
```

7.5.6 Paraphrasing

Our rules allow us to ascribe ontoterms to sentences and phrases. This in turn enables us to define what paraphrasing is. We will cosider two sentences to be paraphrasing each other if both are realizations of the same ontoterm:

```
paraphrase(S1,S2,C):-s(S1,C),s(S2,C).
```

Similarly, a noun phrase could parahrase a sentence if both represent the same ontoterm:

```
paraphrase(S,NP,C):-s(S,C),np(NP,C).
```

```
paraphrase(NP,S,C):-np(NP,C),s(S,C).
```

Obviously, two noun phrases can be paraphrases of each other as well:

```
paraphrase(NP1,NP2,C):-np(NP1,C),np(NP2,C).
```

7.5.7 Modified ontology representation

The modified way of representing intersected concepts forces us to update our ontology definition. The new taxonomy has the following rules:

Anything is under "univ", representing the universal concept:

```
isa(_,univ).
```

`isa` is the transitive closure of the direct subsumption relation `sub`:

```
isa(X,Z):-sub(X,Y),isa(Y,Z).
```

An intersection of some concepts is subsumed by some concept if any of the intersected concepts are subsumed by that concept:

```
isa([X|_T],Z):-isa(X,Z).
isa([_X|T],Z):-isa(T,Z).
```

Due to the changes introduced we store now the taxonomical subsumption information in relation called `sub`:

```
sub(oxygen,substance).
sub(transport,process).
```

7.5.8 Alternative role recognition for prepositional phrases

The definition of the `a_pp` predicate can be refined in order to make use of only one table for the role recognition in prepositional phrases:

```
a_pp(X,P,Y,Z):-t_pp(A,P,B,R),isa(X,A),isa(Y,B),i
    (X,[p(R,Y)],Z).

t_pp(process,of,substance,pnt).
t_pp(influence,of,process,pnt).
t_pp(process,by,univ,agt).
t_pp(transport,across,univ,via).
t_pp(body_part,of,person,loc).
t_pp(property,of,univ,wrt).
t_pp(influence,through,process,bmo).
t_pp(combination,of,univ,cmp).
```

7.5.9 Examples

Let us consider several examples of sentences and noun phrases and their corresponding ontological semantics produced by the system.

The sentence

```
[these,low,insulin,level,inhibit,
 transport,of,glucose,across,cell,membrane]
```

results in the following ontological semantics:

```
[inhibition,
 p(agt,[level,low,p(wrt,[insulin])]),
 p(pnt,[transport,p(pnt,[glucose]),
                   p(via,[membrane,p(loc,
                                     [cell])]
```

The following sentence contains a conjunction at the verb phrase level, which is read distributively, hence resulting in more than one ontological semantics:

```
[low,insulin,level,inhibit,
 transport,of,glucose,
 and,stimulate,transport,of,oxygen]
```

The results corresponding to two verb phrases joined with the conjunction are:

```
[inhibition,
 p(agt,[level,low,p(wrt,[insulin])]),
 p(pnt,[transport,p(pnt,[glucose])])]
```

```
[stimulation,
 p(agt,[level,low,p(wrt,[insulin])]),
 p(pnt,[transport,p(pnt,[oxygen])])]
```

Let us consider the following noun phrase:

```
[increase,of,dna,replication,and,protein,
    synthesis]
```

The conjunction can be regarded as distributive, splitting the phrase into two independent phrases, thus giving rise to the following ontoterms:

```
[increase,p(pnt,[replication,p(pnt,[dna])])]
```

and

```
[synthesis,p(pnt,[protein])]
```

However, the conjunction can also be understood as being embedded within the prepositional phrase, thus giving us one more possible reading:

```
[increase,p(pnt,[synthesis,p(pnt,[protein])])]
```

However, the conjunction can also be understood in one surprising way, i.e. as a distributive connection of "replication" and "protein". It is very hard to visualize such an interpretation, as we know it is not a correct one. Imagine the sentence with the following added parentheses for the purpose of visualizing the scope of the conjunction: "increase of dna (replication and protein) synthesis". Now the rules allow us to apply the distributive reading to the conjunction, hence producing the meaning as if for the following phrase: "increase of dna protein synthesis". Thus, we obtain the following ontological semantics:

```
[increase,p(pnt,[synthesis,p(pnt,[dna]),p(pnt,[
    protein])])]
```

While this result is not desired, it illustrates how largely ambiguous the grammar becomes when introducing conjunctions at all possible levels. Such a problem could be overcome by introducing a larger number of fine–grained grammatical rules to the system.

As a last example, let us see the effect of the collective reading of the conjunction. The sentence

```
[combination,of,insulin,and,c_peptide]
```

produces the desired collective reading:

```
[combination,p(cmp,[c_peptide]),p(cmp,[insulin])
    ]
```

7.6 Definite Clause Grammar Ontograbber

We now present an ontograbber expressed in a special notation available in Prolog system, i.e. Definite Clause Grammars. It improves the performance of the ontograbber due to the fact that it uses difference lists for parsing. In order to view the relation between natural language and its semantics from another perspective, we will use this ontograbber for generating sentences, rather than understanding them.

The biggest difference between this ontograbber and the previous one is the part where the grammar is specified:

```
np([SC]) --> [N],{lex(noun,N,SC)}.

np(C) --> {a_nnc(C1,C2,C)},np(C1),np(C2).

np(C) --> {a_pp(C1,P,C2,C)},np(C1),[P],np(C2).

np(C) --> {a_gen(C1,C2,C)},np(C1),[s],np(C2).

np(C) --> {a_adj(CA,CNP,C)},a(CA),np(CNP).

a([C]) --> [A],{adj(A,C)}.

s(C) --> {a_tv(C1,TVC,C2,C)},np(C1),
         tv(TVC),np(C2).

tv([C]) --> [V],{lex(tv,V,C)}.
```

The intersection of concepts is still represented using a list, except that our new definition rejects empty lists for intersections:

```
i(C1,C2,C):-append(C1,C2,C),
            nonempty(C1),
            nonempty(C2).
nonempty([_|_]).
```

In the `a_nnc` predicate we swap the order of literals in the body:

```
a_nnc(X,Y,Z):-i(Y,[p(R,X)],Z),
              t_nnc(A,B,R),
              isa(X,A),
              isa(Y,B).
```

We make similar operation with the `a_pp` predicate:

```
a_pp([H|T],P,Y,Z)
  :-i([H|T],[p(R,Y)],Z),
    t_pp(A,P,B,R),
    isa_chk([H|T],A),
    isa_chk(Y,B).
```

Again, the same needs to be done with the `a_tv` predicate:

```
a_tv(X,Y,Z,V)
  :-i(W,[p(RO,Z)],V),
    i(Y,[p(RS,X)],W),
    t_tv(A,RS,RO),
    isa(Y,A).
```

The final change is in the way our ontology is defined:

```
isa_reflexive(X,X):-lex(_,_,X).

isa_transitive(X,Y):-sub(X,Y).
isa_transitive(X,Z):-sub(X,Y),
                     isa_transitive(Y,Z).

isa_univ(X,univ):-sub(X,_).
isa_univ(X,univ):-sub(_,X).

isa([X],Y):-isa_reflexive(X,Y).
isa([X],Y):-isa_transitive(X,Y).
isa([X],Y):-isa_univ(X,Y).

isa_chk(X,Y):-member(M,X),isa([M],Y).
```

Now we can find language realizations for a given ontoterm. They can be either sentences or noun phrases:

```
realize(C,L):-s(C,L,[]).
realize(C,L):-np(C,L,[]).
```

In those two definitions the empty list is used due to the fact that Definite Clause Grammars utilize difference lists for parsing, and we do not wish to be left with any part of the sentence unparsed.

7.6.1 Example

Let us consider the following, quite complex concept in the generative ontology.

```
[inhibition,
```

```
p(agt,[low,
        level,
        p(wrt,[insulin])]),
p(pnt,[transport,
        p(pnt,[glucose]),
        p(via,[membrane,
                p(loc,[cell])])])]
```

Let us try to use our system for finding linguistic realizations of it. Below we present the list of produced phrases.

```
[inhibition,by,low,insulin,level,
 of,glucose,transport,across,cell,membrane]

[inhibition,by,low,insulin,level,
 of,transport,of,glucose,across,cell,membrane]

[inhibition,by,low,insulin,s,level,
 of,glucose,transport,across,cell,membrane]

[inhibition,by,low,insulin,s,level,
 of,transport,of,glucose,across,cell,membrane]

[inhibition,by,low,level,of,insulin,
 of,glucose,transport,across,cell,membrane]

[inhibition,by,low,level,of,insulin,
 of,transport,of,glucose,across,cell,membrane]

[low,insulin,level,inhibit,
 glucose,transport,across,cell,membrane]

[low,insulin,level,inhibit,
 transport,of,glucose,across,cell,membrane]

[low,insulin,s,level,inhibit,
 glucose,transport,across,cell,membrane]

[low,insulin,s,level,inhibit,
```

```
 transport , of , glucose , across , cell , membrane]

[low , level , of , insulin , inhibit ,
 glucose , transport , across , cell , membrane]

[low , level , of , insulin , inhibit ,
 transport , of , glucose , across , cell , membrane]
```

As we can observe, the results include various usages of verbs, nouns, prepositional phrases and noun–noun compounds. In other words, the system is able to generate many paraphrases of the same concept. The task of paraphrasing is very important in search situations, where the same concept can appear in texts in various forms. Another issue worth noting is that the generated phrases do not have any preferred result. In some scenarios it could be useful to prefer some phrases rather than others. For instance, if the generated phrase is supposed to be understood by humans, it would be preferable if it was a sentence contained an active verb rather than a long noun phrases consisting of nouns and few prepositions.

7.7 Summary

In this chapter we have presented how ontological semantics can be handled computationally for a fragment of natural language. We have transitioned from a specification in First Order Predicate Logic to an implementation in Prolog. We have presented both ontograbbers that can analyze a text, producing equivalent ontological semantics, as well as generate phrases and sentences given a concept in the generative ontology.

The systems presented in this chapter deal with a restricted fragment of natural language. In the following chapter we will proceed with robust analysis of unrestricted natural language. We will deal with the problem of partiality of understanding, where only parts of a sentence can be analyzed by ontological means.

Chapter 8

The computation of ontosemantics for unrestricted natural language

We now present the algorithm which associates ontological semantics with sentences. The retrieved ontological information takes the form of nodes in the generative ontology represented by so–called 'concept covers', to be described in sect. 8.3. The result of the algoritmic analysis of a sentence is a collection of ontoterms.

The novel contribution of the present approach is that the natural language parsing process is driven primarily by the ontology rather than by grammatical production rules. Similarly to unification grammars, syntactic and semantic constraints are utilized in parallel. However, in our approach, we do not construct a parse tree. Syntax plays a secondary role, taking the form of additional constraints, while ontological affinities play the primary role. The algorithm makes use of a grammar, but it is a simplified, rudimentary grammar covering fragments of English.

The algorithm described below proceeds in a bottom–up fashion, similar to forward chaining, robustly parsing complex sentences which are not fully covered by the simplified grammar. It is reminiscent of the Earley parsing

algorithm (presented in Appendix D on page 213) in that it utilizes a similar dynamic programming technique for backtracking and recomputation avoidance, although it does not make use of top–down constraints due to the lack of a formal grammar for natural language in its entirety.

The algorithm presented goes beyond usual partial and shallow parsing techniques like chunking, which do not provide recursive phrase structure necessary for construction of nodes in the generative ontology.

8.1 Indexing

We index all the ontologically meaningful tokens of the input text. Consider the sample sentence S_1:

"These low_0 insulin_1 levels_2 inhibit_3 the transport_4 of glucose_5 across cell_6 membranes_7."

Only nouns, verbs and adjectives get indexed, as they correspond to some class in the ontology. Nouns, verbs and adjectives are the main building blocks carrying ontological meaning. Even though prepositions do not represent any classes in the ontology, they are utilized by the algorithm, as they usually determine how classes should be combined into compound nodes in the generative ontology.

8.2 Microontology for a sentence

A microontology is the smallest fragment of the generative ontology which relates to the words in a sentence together with inherited ontological affinities.

The microontology for the sentence S_1 is presented below:

c_1 \ c_2	low	insulin	level	inhibition	transport	glucose	cell	membrane
low								
insulin								
level	*CHR*	*WRT*				*WRT*		
inhibition		*AGT*	*AGT*	*PNT*	*PNT*	*AGT*		
transport		*PNT*				*PNT*	*VIA*	*VIA*
glucose								
cell							*LOC*,*POF*	*LOC*,*POF*
membrane							*LOC*,*POF*	*LOC*,*POF*

For each role r given in the table, the concept $c_1[r : c_2]$ is a valid node in the generative ontology. This can be also represented by a graph, see Figure 8.1 on page 144.

8.3 Concept covers

The algorithm manages sets of concept covers (covers for short), which represent associations between the text and nodes in the generative ontology.

Covers are of the following type:

$$\mathbb{N} \times \mathbb{N} \times syn \times N \times P$$

That means, a concept cover is a quintuple of:

- Two numbers m, n, such that $m \leq n$, representing the spanning interval. The integers are indices of the words in the sentence
- The syntactic category representing what kind of word or phrase the cover represents (VP, NP, ...)
- The node in the generative ontology that the cover represents
- The actual phrase covered

As an example, consider the following cover:

$$(4, 5, \text{NP}, transport[PNT : glucose], \text{“transport of glucose”})$$

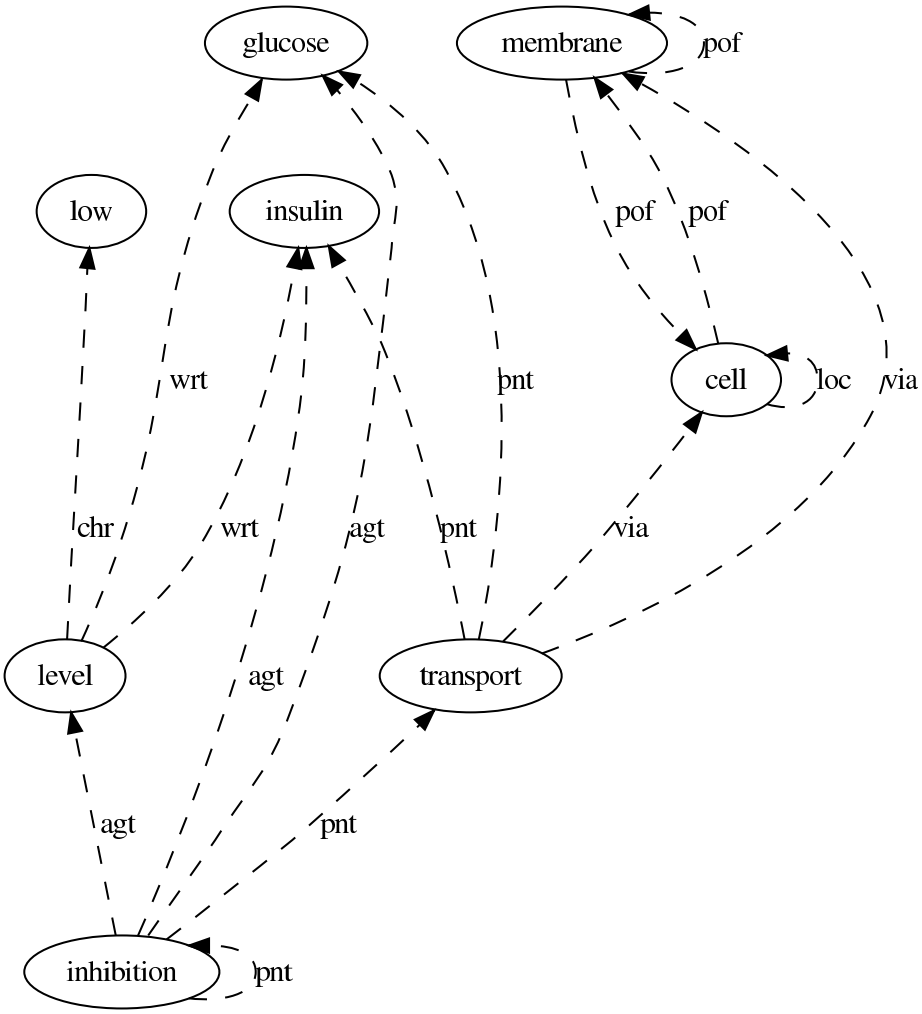


Figure 8.1: The microontology presented in a graph form.

It covers the sentence between positions 4 and 5, it corresponds to a noun phrase, represents the $transport[PNT : glucose]$ node in the generative ontology, and comes from the phrase "transport of glucose".

The iterative algorithm works on a state consisting of two sets: C and H. At each stage of computation, C is the set of all covers found so far. Only covers which are sound with respect to both syntactic and ontological constraints are ever present in C. At any stage of computation H, such that $H \subset C$, is the set of "hot" covers. Hot covers are those that have been found in previous stages and have not yet been "tried" for possible concept combinations. When H is empty, the algorithm stops with C as result. To illustrate, for the sample sentence S_1 we get the following initial set C_0 of concept covers:

$$
\begin{aligned}
C_0 = \{ \quad & (0, 0, \mathrm{A}, \mathit{low}, \text{“low”}), \\
& (1, 1, \mathrm{N}, \mathit{insulin}, \text{“insulin”}), \\
& (2, 2, \mathrm{N}, \mathit{level}, \text{“levels”}), \\
& (3, 3, \mathrm{TV}, \mathit{inhibition}, \text{“inhibit”}), \\
& (4, 4, \mathrm{N}, \mathit{transport}, \text{“transport”}), \\
& (4, 4, \mathrm{TV}, \mathit{transport}, \text{“transport”}), \\
& (5, 5, \mathrm{N}, \mathit{glucose}, \text{“glucose”}), \\
& (6, 6, \mathrm{N}, \mathit{cell}, \text{“cell”}), \\
& (7, 7, \mathrm{N}, \mathit{membrane}, \text{“membranes”})\}
\end{aligned}
$$

Initially, only covers corresponding to single words are present in both sets (beginning of cover interval being the same as the end for each element).

The algorithm begins by having all the initial covers in both sets: $C = C_0$ and $H = C_0$. All covers are in C, because they all represent valid concepts. On the other hand, they are all in H, because all concepts have to be "tried" for possible combinations. In the most pessimistic case, when no concept can be meaningfully combined with any other concept, the initial set C_0 represents a result corresponding to "keyword–level fallback".

Observe that there are two covers in the set corresponding to the word "transport". These have the same cover interval, but different syntactic categories. Eventually, only one of those entries will be utilized for building a large cover. This can be viewed as a part–of–speech tagging process guided by syntactic and ontological constraints.

8.4 Grabbing

The process of grabbing proceeds as follows:

- An arbitrary "hot" concept cover h is selected, such that $h \in H$
- All left–neighbours and right–neighbours of h are found by utilizing h's cover interval information
- A set of pairs P is constructed:

$$P = \{(l, h) | l \text{ is left–neighbour of } h\} \cup \{(h, r) | r \text{ is right–neighbour of } h\}$$

- For each pair $(e_1, e_2) \in P$ we try to combine the covers e_1 and e_2. Combining is explained in sect. 8.5. The result of combining is a set N of new covers. If e_1 and e_2 cannot be combined, $N = \emptyset$. The set of all found covers C is extended to C', so that $C' = C \cup N$. The set of hot covers H is extended to H', so that $H' = (H \cup (N \backslash C)) \backslash \{h\}$. In other words, newly found concepts are reflected in H' if they have not been found before. The just–tried cover h is removed from the hot set
- The process of grabbing continues with C' and H' or terminates when $H' = \emptyset$, that is, nothing is left in the "hot" set. At this point, C' represents the result: all covers, representing found concepts together with their coverage. If the whole sentence was understood, there will be a cover in C' such that it covers the entire sentence. Otherwise, we can select the concept with the biggest coverage as the result.

8.5 Combining explained

Combining two concept covers e_1 (representing concept c_1 and syntactic category syn_1) and e_2 (representing concept c_2 and syntactic category syn_2) proceeds as follows:

- We check whether (and how) concepts can be combined ontologically. A set R of roles r is found using ontological affinities from the microontology, such that $c_1[r : c_2]$ or $c_2[r : c_1]$ is a valid node in the generative ontology. In other words:

$$R = \{r | isa(c_1[r : c_2], \top)\} \cup \{r | isa(c_2[r : c_1], \top)\}$$

- We check how any found combination could be realized syntactically by utilizing syntactic matching rules. Each matching rule gives us a string to be matched against the sentence. Matching rules have the form:

$$syn_1, r, syn_2 \Rightarrow pattern, syn'$$

The elements on the lhs of $\Rightarrow$ represent restrictions on when the rule is applicable. syn' is the syntactic category of the resulting pattern. Some of the matching rules:

$$\begin{aligned} \text{VP}, AGT, \text{NP} &\Rightarrow str2@str1, \text{S} \\ \text{TV}, PNT, \text{NP} &\Rightarrow str1@str2, \text{VP} \\ \text{NP}, CHR, \text{A} &\Rightarrow str2@str1, \text{NP} \end{aligned}$$

As an example, consider the rule:

$$\text{VP}, AGT, \text{NP} \Rightarrow str_2@str_1, \text{S}$$

The rule says that we can combine concept c_1 represented by a verb phrase with concept c_2 represented by a noun phrase using the agent role. The result is a concatenation of strings corresponding to c_2 and c_1, respectively, and the resulting syntactic category is S. The syntactic information could be extracted from this rule into BNF notation:

$$\text{S} ::= \text{NP VP}$$

To illustrate, assume we have two concept covers:

$$e_1 = e(2, 4, \text{VP}, \quad inhibition[PNT : transport[PNT : glucose]], \text{"inhibits glucose transport"})$$

$$e_2 = e(1, 1, \text{NP}, insulin, \text{"insulin"})$$

In this case, we can combine e_1 and e_2 as $c_1[AGT : c_2]$ is a valid node in the generative ontology. Since the syntactic categories fit, we construct the pattern

$$[\text{"}insulin\text{"}, \text{"}inhibits\text{"}, \text{"}glucose\text{"}, \text{"}transport\text{"}]$$

which is successfully matched against the sentence. Hence, we add a new cover to the sets:

$$e(1, 4, \mathrm{S}, \quad inhibition[PNT : transport[PNT : glucose], AGT : insulin], \text{"insulin inhibits glucose transport"})$$

In addition to the matching rules mentioned above, we also have a highly ambiguous rule for noun–noun compounds:

$$\mathrm{N}, \rho, \mathrm{N} \Rightarrow str2@str1, \mathrm{N}$$

In the above, ρ can stand for any role, see also [22]. The high syntactic ambiguity of this rule is usually resolved with the help of ontological affinities.

Prepositions are related to roles by the following many–to–many relation:

Role	Preposition
pnt	"of"
agt	"by"
via	"across"
loc	"of"
wrt	"of"
bmo	"through"
cmp	"of"
⋮	⋮

For each related role ρ and preposition ϕ, we add one rule to our algorithm:

$$\mathrm{NP}, \rho, \mathrm{NP} \Rightarrow str1@[\phi]@str2, \mathrm{NP}$$

In addition, the algorithm utilizes syntactic lifting rules which do not aim at combining concepts but allow the lifting of syntactic categories, e.g., from noun to a noun phrase or from intransitive verb to a verb phrase.

8.6 Grammar

The purely syntactic parsing part of the algorithm could be achieved by full relaxation of ontological affinities, e.g., adding the role *any*:

$$isa(\top[any : \top], \top)$$

"These low insulin levels inhibit the transport of glucose across cell membranes."

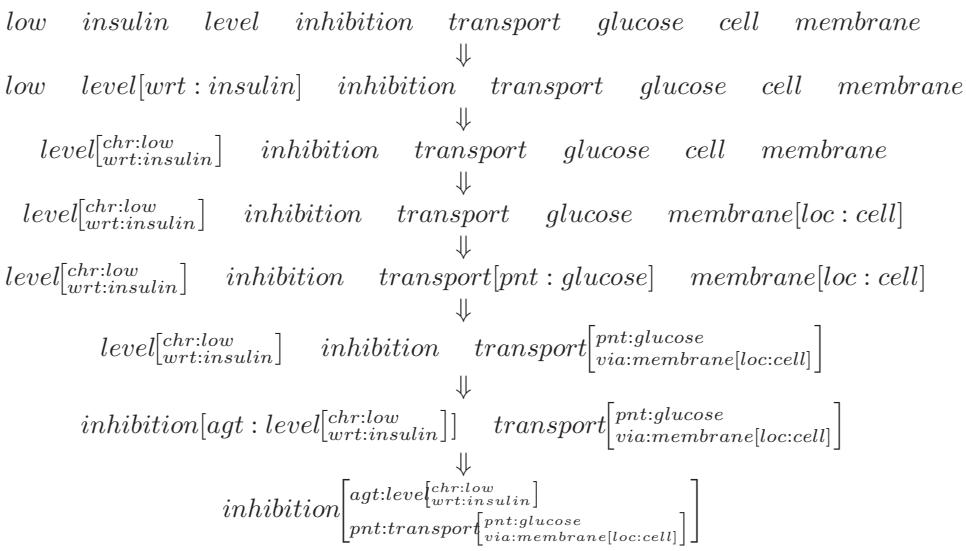


Figure 8.2: Sample run of the ontograbber

In such a case, any concept can be combined with any other from the ontological point of view, effectively making the algorithm use only syntactic constraints.

Since the formal grammar covers only fragments of the language, one cannot, in general, parse the sentence in its entirety. Therefore, we cannot assume any particular starting symbol, e.g., S. In many cases we might only be able to capture a few phrases. In order to model that, we could introduce a modified grammar, with production rules containing a distinguished nonterminal *junk* representing any string. Instead, however, we simply parse the sentence in a bottom–up manner, without any starting symbol.

8.7 Sample run

Figure 8.2 on page 149 presents a sample execution of the ontograbber algorithm. It abstracts from the details of concept covers, and only illustrates in what order concepts are combined and what is the resulting ontological semantics.

8.8 Ontograbber from the parsing perspective

The parsing performed by the algorithm is quite unusual, as it is executed as an additional constraint on top of the ontological affinities. The ontological well-formedness of the ontoterms is the primary driving mechanism of the algorithm.

In order to describe the pure grammatical parsing performed by the algorithm, we could imagine a situation, where ontological constraints are fully relaxed. This is a very unlikely scenario, which constitutes the absolutely worst case. In such a case, any two neighbouring concepts can be combined in a sound manner according to the ontological affinities.

The grammatical constraints take the form of a restricted context free grammar. The restriction is that at most two nonterminals can appear in the right hand side of a production rule. This restriction comes from the fact that the analysis is ontology–driven. The algorithm tries to put together two neighbouring concepts using the affinities, and each of those concepts represents a given grammatical class. Therefore when viewed as a context free grammar, the right hand side of any rule will have at most two nonterminals corresponding to the two concepts being combined.

It is worth observing that the grammar is therefore similar to the CNF. Strictly, our grammar allows more expresive rules. CNF allows rules of the form where there is exactly two nonterminals on the right-hand side, without any terminal symbols. The grammar of our algorithm allows two nonterminals augmented by arbitrary number of terminal symbols. Hence, any valid CNF production rule is also a valid rule in the ontograbber's grammar.

The allowance of an arbitrary number of terminal symbols in the rules is useful for the purpose of writing rules involving closed word classes, e.g. prepositions, determiners, etc.

Due to the fact that any CFG can be rewritten to an equivalent CNF rules, and the fact that CNF rules is a subset of the ontograbber's grammatical rules, we conclude that CFG is not more expressive than our grammar.

The idea of introduction of full CFG production rules has been abandoned. This is because more than two nonterminals on the right hand side of a rule do not correspond well to the way that ontological affinities are defined. It is not clear at present how or why one would want to combine three or more concepts at once. Three concepts can be easily combined in two steps: two of them are combined, followed by the combination of the result with the third one.

Additionally, we are not interested in constructing a new parsing algorithm. There is a number of interesting approaches to parsing, many of which handle full CFG productions. One example could be the Earley parsing algorithm.

One could easily use some of the so-called "of–the–shelf" parsers in order to parse with full CFG. We, however, consider the grammatical constraints to be of secondary importance. We are more interested in the development of a ontological analyzer, rather than grammatical parser.

The fact that full CFG rules are not allowed does not seem to be a large obstacle. There exist also widely used algorithms that limit the grammar to CNF form, e.g. the CKY algorithm. The popularity of this algorithm lets us to believe that in practice the limitation on the form of the rules is not a severe problem.

8.9 Capturing natural language

As we have discussed, the algorithm from grammatical point of view is able to describe any context free language. The problem of capturing a given natural language (or an interesting subset of it) can be reduced to the problem of such a language (or its subset) being a context free language.

This question cannot be answered definitively. Nobody has ever proven one way or the other. The general consensus is that natural languages are context free, although some parts of natural language might not be very conveniently modeled using context–free production rules.

A general problem is that the grammar of natural languages is rarely described using fully formal methods. Natural languages are extremely complicated, constantly evolving, varying across regions. Any two native speakers will at times find a sentence, which one considers grammatical, and the other ungrammatical. For those reasons nobody has ever managed to fully formalize the grammar of any written natural language.

Thus, the question of whether the ontograbber is able to handle any natural language in its entirety boils down to whether the given language is actually context–free.

8.10 Termination

The proof of the termination of the algorithm proceeds as follows:

When two neighbouring concepts are combined the first one covering positions between a and b, the second one between $b+1$ and c, the result will cover positions between a and c. Each cover is non–empty, so the result always covers more than any of its constituents. Hence, the algorithm could not terminate

only in the case if some covers would be growing to unbounded sizes. This is however impossible, as below the index 0 and above the length of the sentence, no neighbours can be found for further combination.

8.11 Complexity issues

The running time and memory complexity of the algorithm is exponential in the worst case. To show that, imagine that the sentence consists of many prepositional phrases, each of which could be either alligned or nested. Additionally imagine that the ontological affinities do not restrict any of those compositions. In such a case we get exponential number of possible parses in the number of prepositional phrases present.

Our algorithm uses the dynamic programming principle in order to avoid the inefficiencies associated with typical top–down parsing, where many trees are rebuilt numerous times while possible rules are tried. In our approach any sound parse tree is kept in memory for further use, so that recomputation of those is not necessary.

Although the worst case running time of the algorithm is exponential, in reality this does not pose a serious problem. Often, there is only a few prepositional phrases present in a sentence, which can be both attached or nested. The ontological affinities filter out many unsound parses. In addition, the sentences are not of unbounded length. Typical sentence might have around 20 words. The algorithm analyses only meaning carrying components of the sentence, so for a sentence of 20 words, around 10 concepts are usually identified. In most cases the amount of ambiguities is not large.

The widely used of–the–shelf parsers need to have extremely good complexities, as they are used e.g. for parsing programming languages, where there might be tens of thousands of terminals present in the input sentence. Earley parser or CKY chart parsers are examples of parsers that work in cubic time. However, when the length of sentences is limited to as few as 20 tokens, we do not need to use such an efficient approach.

8.12 Building ontoterms

The ontograbber builds ontoterms by attaching one ontoterm at the root of another ontoterm using some case role in the process of combining. Such mechanism is enough to build all possible tree–like structured ontoterms, which can

be represented as feature structures.

The limitation of this approach is that it is not possible to build general graph–like ontoterms, that go beyond trees in their shape.

8.13 Applied generative ontology for bio–domain

In Figure 8.3 on page 154 we present a sample top ontology taxonomy that could be used as the input to the ontograbbing algorithm.

Similarly, the following ontological affinities could be used as the input to the algorithm, coupled with the top ontology presented in Figure 8.3.

anatomical_structure	pof	anatomical_structure
influence	agt	substance
influence	agt	level
influence	pnt	phenomenon_or_process
physical_object	loc	anatomical_structure
physical_object	pof	physical_object
production	agt	anatomical_structure
production	pnt	substance
substance_portion	wrt	substance
transport	pnt	substance
transport	via	anatomical_structure
level	chr	position_on_scale
level	wrt	substance
⋮	⋮	⋮

8.14 Real life examples

Let us now perform some experiments concerning how the algorithm presented in this chapter performs on some sample sentences. Below we present a list of sentences, which is a random sample from large biomedical corpora. Out of millions of sentences these specific sentences have been chosen only according to how much of them is covered by our microscopic test lexicon. The lexicon is so small that it was difficult to find a large number of sentences to experiment with. Those presented here have not been preselected according to how well they are handled by the algorithm. For each sentence we first give a set of three idealistic ontological semantics, which have been created by hand by three experts:

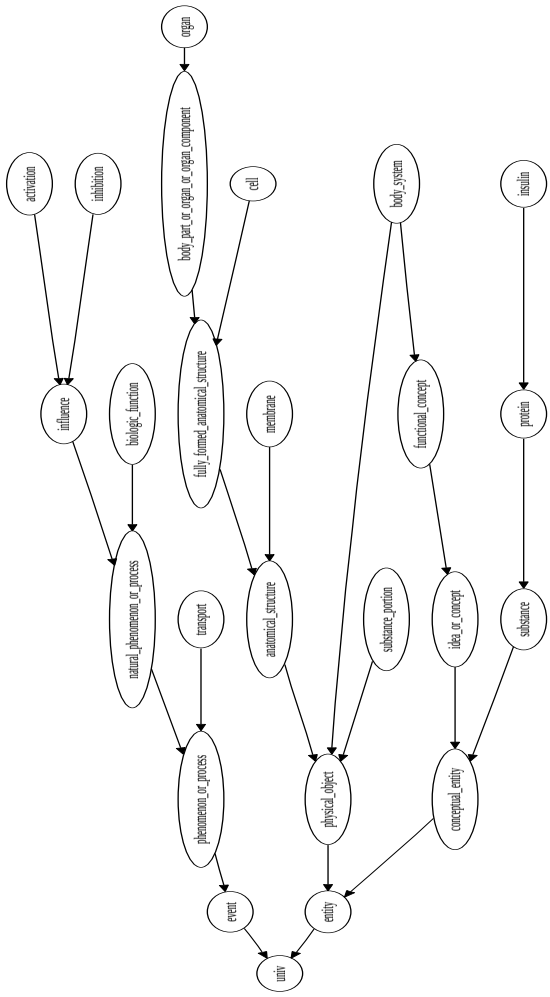


Figure 8.3: Sample top ontology taxonomy to be used as the input to the ontograbbing algorithm

- Bartlomiej Antoni Szymczak (abbreviated BAS) from Technical University of Denmark
- Tine Lassen (abbreviated TL) from Roskilde University, Denmark
- Hanne Erdman Thomsen (abbreviated HET) from Copenhagen Business School, Denmark

All experts are members of the SIABO project and specialize in the use of generative ontologies and ontological semantics.

The semantics prepared by experts is believed to be a "perfect answer", which one would ideally like to get as the output of the algorithm. In addition to those golden ontoterms we also list the ontoterm(s) produced by the algorithm. Due to ambiguities, there is often more than one ontosemantics produced (both by experts and the algorithm).

Unfortunately, producing ontological semantics by hand is difficult and time consuming, and in addition requires expert knowledge. Therefore we did not have resources to acquire hand–produced ontosemantics for more than ten sentences. We have rejected the option of producing golden standard semantics exclusively ourselves, as it is difficult to create such semantics objectively while knowing how the algorithm operates and what can be expected of it.

We have performed tests on 37 realistic sentences. Out of these we first present 10 for which we managed to get experts' golden standard. Next, we present the remaining 27 sentences, for which we do not have a golden standard. Adding a single sentence to the test set is not trivial, as both the lexicon and the generative ontology need to be manually extended to incorporate the new words and concepts.

The input sentence 1:

```
these low insulin levels inhibit the transport of glucose
    across cell membranes therefore causing high blood
   glucose levels
```

Golden ontosemantics (BAS):

```
causation [agt:inhibition[agt:[level[wrt:insulin,
                                     chr:low]],
                          pnt:transport[pnt:glucose,
```

```
                                                    via:membrane[pof:
                                                        cell]]],
5                   pnt:level[chr:high,
                              wrt:glucose[loc:blood]]]
```

Golden ontosemantics (TL):

```
inhibition[AGT:level[WRT:insulin,CHR:low],PNT:transport[
    WRT:glucose[WRT:
membrane[WRT:cell]]],RST:level[WRT:glucose[LOC:blood],CHR
    :high]]
```

Golden ontosemantics (HET):

```
inhibition [AGT:[[level [WRT: insulin]] CHR: low];
            PNT: transport [PNT: glucose; VIA: membrane [
                LOC: cell];
            CAUS: [[[level [WRT: glucose [LOC: blood]]
                CHR: high]
```

Ontograbber's tagged sentence:

```
these[] low[A] insulin[N] levels[N] inhibit[TV] the[]
    transport[N TV] of[] glucose[N] across[] cell[N]
    membranes[N] therefore[] causing[VBG] high[A] blood[N]
     glucose[N] levels[N]
```

Ontograbber's output ontosemantics:

- (0,7,S,inhibition $\begin{bmatrix} agt: \text{level}\begin{bmatrix} chr: \text{low} \\ wrt: \text{insulin} \end{bmatrix} \\ pnt: \text{transport}\begin{bmatrix} pnt: \text{glucose} \\ via: \text{membrane}[loc: \text{cell}] \end{bmatrix} \end{bmatrix}$,"low insulin levels inhibit transport of glucose across cell membranes")
- (0,7,S,inhibition $\begin{bmatrix} agt: \text{level}\begin{bmatrix} chr: \text{low} \\ wrt: \text{insulin} \end{bmatrix} \\ pnt: \text{transport}\begin{bmatrix} pnt: \text{glucose} \\ via: \text{membrane}[pof: \text{cell}] \end{bmatrix} \end{bmatrix}$,"low insulin levels inhibit transport of glucose across cell membranes")

- (8,8,VBG,causing,"causing")
- (9,12,NP,level $\begin{bmatrix} chr\text{: high} \\ wrt\text{: glucose} \begin{bmatrix} src\text{: blood} \end{bmatrix} \end{bmatrix}$,"high blood glucose levels")
- (9,12,NP,level $\begin{bmatrix} chr\text{: high} \\ loc\text{: blood} \\ wrt\text{: glucose} \end{bmatrix}$,"high blood glucose levels")

Discussion:

This is a sentence where all experts have agreed to a large extent on the golden ontological semantics. They have in addition all decided that the semantics should be represented by one very complex concept. The algorithm was mostly capable of understanding the sentence. It has failed to merge to large ontoterms into one covering entire sentence, as it was unable to understand the formulation 'therefore causing'. Also, some small ambiguities are visible in the result, i.e. the treatment of 'cell membranes', which could be either membranes located in cells or membranes being part of cells. Also, 'blood glucose levels' is a noun–noun–noun compound that is ambiguous to the algorithm.

The input sentence 2:

```
glucose enters the betacells through the glucose
    transporter glut2
```

Golden ontosemantics (BAS):

```
1. entering[agt:glucose,pnt:beta_cell,via:glut2]
2. being[agt:glut2,pnt:transporter[wrt:glucose]]
```

Golden ontosemantics (TL):

```
entrance[THM:glucose,LOC:betacell,PATH:glut2[CHR:
    transporter[WRT:glucose]]]
```

Golden ontosemantics (HET):

```
entrance          [AGT: glucose;
        GOAL: betacell;
        VIA: transporter [PNT: glucose];
```

```
                VIA: glut2]
```

Ontograbber's tagged sentence:

```
glucose[N] enters[TV] the[] betacells[N] through[] the[]
    glucose[N] transporter[N] glut2[N]
```

Ontograbber's output ontosemantics:

- (0,5,S,entry $\begin{bmatrix} agt\text{: glucose} \\ pnt\text{: betacell} \\ via\text{: glut2}\left[wrt\text{: transporter}\left[wrt\text{: glucose}\right]\right] \end{bmatrix}$, "glucose enters betacells through glucose transporter glut2")
- (0,5,S,entry $\begin{bmatrix} agt\text{: glucose} \\ pnt\text{: betacell} \\ via\text{: glut2}\begin{bmatrix} wrt\text{: glucose} \\ wrt\text{: transporter} \end{bmatrix} \end{bmatrix}$, "glucose enters betacells through glucose transporter glut2")

Discussion:

In this sentence we can observe that the experts have used different roles for constructing golden semantics. They had full freedom as far as the choice of atomic concepts and roles is concerned. Also notice that the sentence contains the phrase 'glucose transporter glut2', which is very difficult to analyze without domain–specific expert knowledge. The experts BAS, TL and HET specialize in ontological semantics, however they cannot be regarded as experts in biology or medical domain. It is meaningless to judge whether the algorithm's treatment of 'glucose transporter glut2' is correct, as the experts couldn't easily agree on it.

The input sentence 3:

```
insulin also inhibits the release of glucose from the
    liver
```

Golden ontosemantics (BAS):

```
inhibition [agt:insulin ,
            pnt:release [pnt:glucose ,src:liver ]]
```

Golden ontosemantics (TL):

```
inhibition [AGT:insulin ,PNT:release [WRT:glucose [SRC:liver
    ]]]
```

Golden ontosemantics (HET):

```
inhibition [AGT: insulin;
        PNT: release [PNT: glucose [SRC: liver ]]]
```

Ontograbber's tagged sentence:

```
insulin [N] also [] inhibits [TV] the [] release [N] of []
    glucose [N] from [] the [] liver [N]
```

Ontograbber's output ontosemantics:

- (0,4,S,inhibition $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: release} \begin{bmatrix} pnt\text{: glucose} \\ src\text{: liver} \end{bmatrix} \end{bmatrix}$, "insulin inhibits release of glucose from liver")
- (0,4,S,inhibition $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: release} \left[pnt\text{: glucose} \left[src\text{: liver} \right] \right] \end{bmatrix}$, "insulin inhibits release of glucose from liver")

Discussion:

This is a relatively simple sentence, which uses only the most common linguistic constructs. The experts were in agreement as to what ontosemantics should be produced. The algorithm has also successfully extracted the golden semantics. However, it also produced one more result stemming from the ambiguity of the phrase 'release of glucose from liver'. It has found an extra, slightly surprising meaning, where it is 'glucose from liver' that is released, but the release process itself is not necessarily sourced in liver. Humans are very good at rejecting such an alternative. In fact, they reject it so well, that being presented with such an answer appears to be surprising.

The input sentence 4:

```
the secreted insulin promotes glucose utilization and
    inhibits production of glucose by the liver
```

Golden ontosemantics (BAS):

```
1. promotion[agt:insulin[chr:secreted],
             pnt:utilization[pnt:glucose]]
2.inhibition[agt:insulin[chr:secreted],
             pnt:production[agt:liver,pnt:glucose]]
```

Golden ontosemantics (TL):

```
promotion[AGT:insulin[CHR:secreted],PNT:utilization[WRT:
    glucose]]
inhibition[AGT:insulin[CHR:secreted],PNT:production[AGT:
    liver,RST:glucose]]
```

Golden ontosemantics (HET):

```
[promotion [AGT: insulin [CHR: secretion];
            PNT: utilization [PNT: glucose]];
inhibition [AGT: insulin [CHR: secretion];
        PNT: production [AGT: liver; PNT: glucose]]]
```

Ontograbber's tagged sentence:

```
the[] secreted[VBD VBN] insulin[N] promotes[TV] glucose[N
    ] utilization[N] and[] inhibits[TV] production[N] of[]
     glucose[N] by[] the[] liver[N]
```

Ontograbber's output ontosemantics:

- (0,8,S,promotion $\begin{bmatrix} agt\text{: insulin}\left[pntof\text{: secretion}\right] \\ pnt\text{: utilization}\left[pnt\text{: glucose}\right] \end{bmatrix}$, " secreted insulin promotes glucose utilization and inhibits production of glucose by liver")

- (0,8,S,inhibition $\begin{bmatrix} agt\text{: insulin} \begin{bmatrix} pntof\text{: secretion} \end{bmatrix} \\ pnt\text{: production} \begin{bmatrix} agt\text{: liver} \\ pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$, " secreted insulin promotes glucose utilization and inhibits production of glucose by liver")

Discussion:

In this sentence we encounter the 'and' conjunction at the verb phrase level. The reading of it is distributive, as illustrated by the golden ontosemantics. All experts agree about what the semantics should be, and all have hand–produced two ontoterms corresponding to the distributive reading of the conjunction. The algorithm performs very well for this sentence, understanding the distributivity and producing two correct readings virtually identical to those suggested by experts.

The input sentence 5:

```
three insulin - signaling pathway - specific inhibitors
    also abolish pgg - induced glucose transport in 3t3-l1
     adipocytes
```

Golden ontosemantics (BAS):

```
abolition[agt:inhibitor,
          pnt:transport[wrt:glucose,chr:induced[agt:pgg
              ]]]
```

Golden ontosemantics (TL):

```
abolition[AGT:inhibitor[CHR:specificity[WRT:pathway[WRT:
    signaling[WRT:insulin]]]],PNT:transport[PNT:glucose[
    CHR:induced[WRT:pgg]]],LOC:adipocytes[CHR:3t3-l1]]
or
abolition[AGT:inhibitor[CHR:specificity[WRT:pathway[WRT:
    signaling[WRT:insulin]]]],PNT:transport[PNT:glucose[
    CHR:induced[WRT:pgg]],LOC:adipocytes[CHR:3t3-l1]]]
```

Golden ontosemantics (HET):

```
abolition          [AGT: inhibitor [CHR: signaling [PNT:
    insulin];
```

```
                    CHR: pathway-specific];
3       PNT: transport [PNT: glucose;
      CHR: induction [AGT: pgg];
      LOC: adipocytes [CHR: 3t3-l1]]
```

Ontograbber's tagged sentence:

```
three[] insulin[N] -[] signaling[VBG] pathway[N] -[]
    specific[A] inhibitors[N] also[] abolish[TV] pgg[N]
    -[] induced[VBD VBN] glucose[N] transport[N TV] in[] 3
    t3-l1[N] adipocytes[N]
```

Ontograbber's output ontosemantics:

- (0,1,VBG,signaling[*pnt*: insulin],"insulin - signaling")
- (1,4,NP,signaling[*pnt*: inhibitor[*chr*: specific[*wrt*: pathway]]],"signaling pathway - specific inhibitors")
- (2,11,S,abolition $\begin{bmatrix} \textit{agt}\text{: inhibitor}\left[\textit{chr}\text{: specific}\left[\textit{wrt}\text{: pathway}\right]\right] \\ \textit{loc}\text{: adipocyte}\left[\textit{pof}\text{: q-3t3l1}\right] \\ \textit{pnt}\text{: transport}\begin{bmatrix}\textit{pnt}\text{: glucose} \\ \textit{pntof}\text{: induction}\left[\textit{agt}\text{: pgg}\right]\end{bmatrix}\end{bmatrix}$,"pathway - specific inhibitors abolish pgg - induced glucose transport in 3t3-l1 adipocytes")
- (2,11,S,abolition $\begin{bmatrix} \textit{agt}\text{: inhibitor}\left[\textit{chr}\text{: specific}\left[\textit{wrt}\text{: pathway}\right]\right] \\ \textit{loc}\text{: adipocyte}\left[\textit{loc}\text{: q-3t3l1}\right] \\ \textit{pnt}\text{: transport}\begin{bmatrix}\textit{pnt}\text{: glucose} \\ \textit{pntof}\text{: induction}\left[\textit{agt}\text{: pgg}\right]\end{bmatrix}\end{bmatrix}$,"pathway - specific inhibitors abolish pgg - induced glucose transport in 3t3-l1 adipocytes")
- (2,11,S,abolition $\begin{bmatrix} \textit{agt}\text{: inhibitor}\left[\textit{chr}\text{: specific}\left[\textit{wrt}\text{: pathway}\right]\right] \\ \textit{pnt}\text{: transport}\begin{bmatrix}\textit{loc}\text{: adipocyte}\left[\textit{loc}\text{: q-3t3l1}\right] \\ \textit{pnt}\text{: glucose} \\ \textit{pntof}\text{: induction}\left[\textit{agt}\text{: pgg}\right]\end{bmatrix}\end{bmatrix}$,"pathway - specific inhibitors abolish pgg - induced glucose transport in 3t3-l1 adipocytes")

- (2,11,S,abolition $\begin{bmatrix} agt\text{: inhibitor}\big[chr\text{: specific}\big[wrt\text{: pathway}\big]\big] \\ pnt\text{: transport}\begin{bmatrix} loc\text{: adipocyte}\big[pof\text{: q-3t3l1}\big] \\ pnt\text{: glucose} \\ pntof\text{: induction}\big[agt\text{: pgg}\big] \end{bmatrix} \end{bmatrix}$, "pathway - specific inhibitors abolish pgg - induced glucose transport in 3t3-l1 adipocytes")

Discussion:

This sentence proves to be very complicated. It requires a biological expert knowledge to understand. The experts have suggested quite different golden semantics for this sentence. There seems to be a lot of disagreement about what the sentence means and how small ontoterms should be combined to provide an ontological meaning for the entire sentence. Hence, it is very difficult to judge how successful the algorithm was, however it seems to be closest to the golden standard suggested by HET.

The input sentence 6:

```
some anti-fungal agents function as cell wall inhibitors
    by inhibiting glucose synthase
```

Golden ontosemantics (BAS):

```
inhibition[agt:agent[chr:anti-fungal],
           wrt:wall[pof:cell],
           pnt:synthase[pnt:glucose]]
```

Golden ontosemantics (TL):

```
function[THM:agent[CHR:anti-fungal],ATT:inhibitor[PNT:
    wall[WRT:cell]],MNR:inhibition[PNT:synthase[WRT:
    glucose]]]
```

Golden ontosemantics (HET):

```
function [AGT: agent [CHR: anti-fungal]
            THM: inhibition [AGT: inhibitor [XXX: wall [
                POF: cell]];
                 PNT: synthase [PNT: glucose]]]
```

Ontograbber's tagged sentence:

```
some[] anti-fungal[A] agents[N] function[N TV] as[] cell[
    N] wall[N] inhibitors[N] by[] inhibiting[VBG] glucose[
    N] synthase[N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,agent$\left[\textit{chr}\text{: antifungal}\right]$,“ anti-fungal agents”)
- (2,2,TV,functioning,“function”)
- (2,2,NP,function,“function”)
- (3,5,NP,inhibitor$\left[\textit{src}\text{: wall}\left[\textit{pof}\text{: cell}\right]\right]$,“cell wall inhibitors”)
- (3,5,NP,inhibitor$\left[\textit{src}\text{: wall}\left[\textit{loc}\text{: cell}\right]\right]$,“cell wall inhibitors”)
- (3,5,NP,inhibitor$\begin{bmatrix}\textit{src}\text{: cell}\\ \textit{src}\text{: wall}\end{bmatrix}$,“cell wall inhibitors”)
- (6,8,NP,inhibition$\left[\textit{pnt}\text{: synthase}\left[\textit{pnt}\text{: glucose}\right]\right]$,“inhibiting glucose synthase”)

Discussion:

This sentence also appears to be very difficult to understand, as the experts have quite different views on what ontosemantics should be extracted ideally. They have however all constructed one large ontoterm, while the algorithm has only managed to construct several smaller ones without merging them into one large. However, it is interesting to observe that the small ontoterms produced by the algorithm all appear as constituents of the large ontoterms produced by experts. It could be argued that these small ontoterms represent the common agreement level between the experts.

The input sentence 7:

```
although insulin secretion is predominantly controlled by
    blood levels of glucose , somatostatin inhibits
   glucose - mediated insulin secretory responses
```

Golden ontosemantics (BAS):

```
1. control[agt:level[wrt:glucose[loc:blood]],
           pnt:secretion[pnt:insulin]]

2. inhibition[agt:somatostatin,pnt:response[??]]
```

Golden ontosemantics (TL):

```
control[AGT:level[WRT:glucose,LOC:blood],PNT:secretion[
    WRT:insulin]
inhibition[AGT:somatostatin,PNT:response[CHR:secretory[
    WRT:insulin],CHR:mediated]]
```

Golden ontosemantics (HET):

```
control [AGT: level [WRT: glucose; LOC: blood];
        PNT: secretion [THM: insulin]
inhibition [AGT: somatostatin;
        PNT: response [BMO: secretion [CHR: mediation [
            AGT: glucose];
             THM: insulin]]
```

Ontograbber's tagged sentence:

```
although[] insulin[N] secretion[N] is[IS] predominantly[]
    controlled[VBD VBN] by[] blood[N] levels[N] of[]
    glucose[N] ,[,] somatostatin[N] inhibits[TV] glucose[N
    ] -[] mediated[VBD VBN] insulin[N] secretory[A]
    responses[N]
```

Ontograbber's output ontosemantics:

- (0,6,PSV,control $\begin{bmatrix} agt\text{: level} \begin{bmatrix} loc\text{: blood} \\ wrt\text{: glucose} \end{bmatrix} \\ pnt\text{: secretion} \begin{bmatrix} pnt\text{: insulin} \end{bmatrix} \end{bmatrix}$, "insulin secretion is controlled by blood levels of glucose")
- (7,7,,,nil,",")
- (8,8,NP,somatostatin,"somatostatin")

- (9,9,TV,inhibition,"inhibits")
- (10,10,NP,glucose,"glucose")
- (11,11,VBD,mediation,"mediated")
- (11,11,VBN,mediation,"mediated")
- (12,12,NP,insulin,"insulin")
- (13,14,NP,response[*chr*: secretory],"secretory responses")

Discussion:

The experts seem to agree about the fact that the ontosemantics should be split into two ontoterms, one corresponding to a large concept of 'control' and the other to a large concept of 'inhibition'. However, the experts had trouble understanding or agreeing upon how to represent the meaning of the phrase 'inhibits glucose–mediated insulin secretory responses'. The algorithm in fact reflects this difficulty, as it has found a satisfactory answer for the 'control' part, but did not manage to go far beyond keywords for the 'inhibition' part.

The input sentence 8:

```
the blood glucose levels may be fasting or fed glucose
    levels , and blood glucose levels include serum or
    plasma glucose levels
```

Golden ontosemantics (BAS):

```
level[wrt:glucose[loc:blood]]
level[wrt:glucose[loc:blood,chr:fed]]
level[wrt:glucose[loc:blood,chr:fasting]]
level[wrt:glucose[loc:serum]]
level[wrt:glucose[loc:plasma]]
```

Golden ontosemantics (TL):

```
being[THM:level[WRT:glucose,LOC:blood], ATT:level[WRT:
    glucose,CHR:fasting], ATT:level[WRT:glucose,CHR:fed]]
being[THM:level[WRT:glucose,LOC:blood], ATT:level[WRT:
    glucose,LOC:serum], ATT:level[WRT:glucose,LOC:plasma]]
```

Golden ontosemantics (HET):

```
level [WRT: glucose; LOC: blood]
level [WRT: glucose; CHR: fasting]
level [WRT: glucose; CHR: feeding]
inclusion        [AGT: level [WRT: glucose; LOC: blood]
        PNT: level [WRT: serum];
        PNT: level [WRT: glucose [CHR: plasma]]]
```

Ontograbber's tagged sentence:

```
the [] blood [N] glucose [N] levels [N] may [CAN] be [BE]
    fasting [VBG] or [] fed [VBD VBN] glucose [N] levels [N]
    ,[,] and [] blood [N] glucose [N] levels [N] include [TV]
    serum [N] or [] plasma [N] glucose [N] levels [N]
```

Ontograbber's output ontosemantics:

- (0,2,NP,level$\left[\textit{wrt}\text{: glucose}\left[\textit{src}\text{: blood}\right]\right]$,“ blood glucose levels”)
- (0,2,NP,level$\begin{bmatrix}\textit{loc}\text{: blood}\\ \textit{wrt}\text{: glucose}\end{bmatrix}$,“ blood glucose levels”)
- (3,3,CAN,can,“may”)
- (4,4,BE,nil,“be”)
- (5,5,VBG,fasting,“fasting”)
- (6,6,VBD,feeding,“fed”)
- (6,6,VBN,feeding,“fed”)
- (7,8,NP,level$\left[\textit{wrt}\text{: glucose}\right]$,“glucose levels”)
- (9,9,,,nil,“,”)
- (10,12,NP,level$\left[\textit{wrt}\text{: glucose}\left[\textit{src}\text{: blood}\right]\right]$,“ blood glucose levels”)
- (10,12,NP,level$\begin{bmatrix}\textit{loc}\text{: blood}\\ \textit{wrt}\text{: glucose}\end{bmatrix}$,“ blood glucose levels”)

- (13,13,TV,inclusion,"include")
- (14,14,NP,serum,"serum")
- (15,15,NP,plasma,"plasma")
- (16,17,NP,level[*wrt*: glucose],"glucose levels")

Discussion:

The sentence is quite difficult to understand, and the experts do not have a common view of what the semantics should be. This is probably due to relatively unusual nature of the sentence. Rather than providing describing observations, it provides a classification. This can be understood as the definition of the biomedical ontology itself, which the algorithm uses as its input. The algorithm performs quite well, as it captures most of the ontoterms common to all those suggested by experts.

The input sentence 9:

```
the insulin secretion determined is preferably glucose -
    stimulated insulin secretion
```

Golden ontosemantics (BAS):

```
secretion[pnt:insulin,chr:stimulated[agt:glucose]]
```

Golden ontosemantics (TL):

```
finding[THM:secretion[PNT:insulin],ATT:secretion[PNT:
    insulin,CHR:stimulation[AGT:glucose]]]
```

Golden ontosemantics (HET):

```
secretion [THM: insulin; CHR: determination]
secretion [THM: insulin; CHR: stimulation [AGT: glucose]]
```

Ontograbber's tagged sentence:

```
the[] insulin[N] secretion[N] determined[VBD VBN] is[IS]
    preferably[] glucose[N] -[] stimulated[VBD VBN]
    insulin[N] secretion[N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,secretion$\left[\textit{pnt}\text{: insulin}\right]$,“ insulin secretion”)
- (2,2,VBD,determining,“determined”)
- (2,2,VBN,determining,“determined”)
- (3,5,PSV,stimulation$\left[\textit{agt}\text{: glucose}\right]$,“is glucose - stimulated”)
- (4,7,NP,secretion$\begin{bmatrix}\textit{pnt}\text{: insulin}\\ \textit{pntof}\text{: stimulation}\left[\textit{agt}\text{: glucose}\right]\end{bmatrix}$,“glucose - stimulated insulin secretion”)

Discussion:

There was a medium–level agreement between the experts on this sentence. One expert suggested two ontoterms, while other suggested one. The algorithm was however unable to find any ontoterm covering entire sentence. It found several ontoterms covering some parts of the sentence. It is worth noticing that two of these overlap, one representing the phrase ‘is glucose – stimulated’, and the other ‘glucose – stimulated insulin secretion’.

The input sentence 10:

```
in the liver , insulin inhibits the production of glucose
    by inhibiting gluconeogenesis and glycogenolysis
```

Golden ontosemantics (BAS):

```
causation[agt:inhibition[agt:insulin,
                         pnt:gluconeogenesis,pnt:
                             glycogenolysis]
          pnt:inhibition[agt:insulin,loc:liver,
                         pnt:production[pnt:glucose]]]
```

Golden ontosemantics (TL):

```
inhibition[AGT:insulin, PNT:production[PNT:glucose], MNR:
    inhibition[PNT:gluconeogenesis,PNT:glycogenolysis],LOC
    :liver]
```

Golden ontosemantics (HET):

```
inhibition [AGT: insulin;
        PNT: production [PNT: glucose];
        BMO: inhibition [PNT: gluconeogenesis; PNT:
            glycogenolysis];
        LOC: liver]
```

Ontograbber's tagged sentence:

```
in[] the[] liver[N] ,[,] insulin[N] inhibits[TV] the[]
    production[N] of[] glucose[N] by[] inhibiting[VBG]
    gluconeogenesis[N] and[] glycogenolysis[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,liver,“ liver”)
- (1,1,,,nil,“,”)
- (2,5,S,inhibition $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: production}\left[pnt\text{: glucose}\right] \end{bmatrix}$,“insulin inhibits production of glucose”)
- (6,8,NP,inhibition$\left[pnt\text{: glycogenolysis}\right]$,“inhibiting gluconeogenesis and glycogenolysis”)
- (6,8,NP,inhibition$\left[pnt\text{: gluconeogenesis}\right]$,“inhibiting gluconeogenesis and glycogenolysis”)
- (6,8,NP,glycogenolysis,“inhibiting gluconeogenesis and glycogenolysis”)

Discussion:

In this sentence the experts do not seem to agree what the ontosemantics should be. Thay have all produced quite different large ontoterms. However, it is worth observing that these large ontoterms, while different, have some small common constituents. The algorithm has captured multiple ontoterms that overlap with these constituents. It also had a surprising approach as to how the conjunction should be interpreted.

8.15 Conclusion

The result of testing the remaining 27 sentences is presented in Appendix I on page 269. The ten results presented and discussed here are of highest importance, as we have been able to provide golden standard ontosemantics produced by three experts.

We believe that the results are very promising. The ontograbber was able to analyze sentences for which it does not know a full grammar. Furthermore, the lexicon used as input to the ontograbber was tiny. In most sentences not all the words have been found in the lexicon. However, despite those shortcomings, the system has managed to extract some ontological semantics for the sentences. The semantics in many cases is quite far from the golden one specified by experts by hand. However, it is worth keeping in mind that the golden semantics assumes perfect lexical and grammatical coverage of each sentence. With the tiny grammar employed in the system and the minute lexicon and ontology, we believe that the extracted semantics is very promsing. It would certainly get much closer to the golden one if the lexicon, ontology, and grammar given as input to the ontograbber would be extended.

In some cases the algorithm failed at merging middle–sized ontoterms into one very large ontoterm covering the meaning of entire sentence. This most often stems from the fact that non–domain specific words are used as connective entities for such meaning or linguistic constructs not covered by our small grammar are used. However, we regard this not to be a major problem, as the middle–sized ontoterms (say, covering half of the sentence each) are sound meaning representations, representing correctly the meaning of some phrases.

It is also important that in all sentences the resulting semantics provides much deeper understanding of meaning content than keyword–based search do. This hints that the resulting semantics could be used for constructing a successful search engine, at least for the domain of biomedical scientific texts.

Chapter 9

Further work

9.1 Incorporation of a large scale lexicon and ontology

All of the ontograbbers presented in Chapters 7 and 8 use two main resources as input, i.e. the lexicon and the ontology. It would be very interesting to use some large scale resources for that purpose, in an attempt to capture as much of the natural language as possible. However, doing so poses many challenges described below, and hence is beyond the scope of the current project.

The ontologies available today do not exhibit the generative character, necessary for our ontograbbers. Hence, they would need to be tailored manually in order to fit as the input to our systems. This generativity could be achieved by providing ontological affinities on the already–existing large taxonomies, in effect creating a large generative ontology. The amount of work here would not be as large as might be suspected, due to the fact that we only need to provide the affinities on the very top level of our ontology and allow the mechanics of the generative ontology to inherit the affinities deeper down. For an example how this is working in practice, see Figure 4.3 on page 82.

Adaptation of a large lexicon as the input to our ontograbber might prove more challenging. The lexicon needs to provide two main pieces of information to our ontograbbers, i.e. the possible parts of speech for every word and what node in the ontology the word actually represents. Even though a large number of lexicons exist which provide the required information on word's parts of speech[1],

[1] Most dictionaries do so.

none of them is immediately ready for use with the generative ontology by providing the information how words link to it. This could be worked around by using some of the lexicons available together with large ontologies available today. Unfortunately, by following this path we would only get simple links of words to the ontology, i.e. linking only to the basic concepts. Ideally, the lexicon would need to be modified to introduce the links of some words to complex ontoterms, which might require a substantial amount of work.

9.2 Extending natural language coverage

One interesting extension of most of the ontograbbers presented in Chapters 7 and 8 is enlarging the grammar of those systems, in order to enable them to capture more natural language constructs. This could be achieved fairly simply for many types of constructs, by following the same principles as with the constructs already taken care of. For instance, the introduction of relative clauses would be based on the same ideas as verb phrases and noun–noun compounds. There are also some constructs, which we have purposedly decided to omit, as their contribution to the ontological semantics we search for is marginal. These include for instance determiners. However, there are some other interesting linguistic constructs that might be more difficult to introduce. An example of that are phrasal verbs, where part of the verb is moved to the end of the sentence. Taking care of that would require some changes in the grammar of the ontograbbers. We also have not discussed how to modify the methods presented in order to deal with languages other than English.

9.3 Ontological search engine

While the problem of ontograbbing is quite well suited for formal description, which does not depart much from the logical specification, it is still important to keep in mind that one wants to perform search based on the assigned ontological semantics. Hence, there is a need for coming up with a mechanism for searching huge amounts of text.

The semantics extracted from the text could be used in a search engine. As a matter of fact, the design was guided with such a possibility in mind. The semantics has been tailored, so that the search can be performed efficiently and easily.

Consider the following semantics extracted from the text:

$$\begin{aligned} &forcing \cap agt : insulin \\ \cap &pnt : (storage \cap pnt : glycogen \\ &\quad \cap loc : (cell \cap wrt : liver)) \end{aligned}$$

Semantics of this form could be pre–extracted from multiple articles and stored in a database, which could be accessed by the search engine. If the user issues a query, we would simply perform the semantic analysis on it, and extract the query semantics. Let us assume that the user is interested in the fact that insulin causes the storage of some substance and issues the following query:

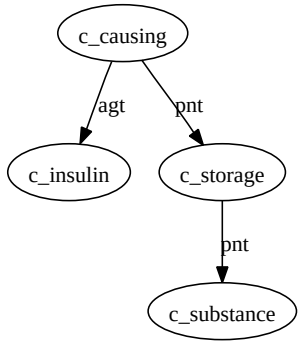
Insulin caused storage of substance. (9.1)

This query would get the following semantics captured:

$$\begin{aligned} &causing \cap agt : insulin \\ \cap &pnt : (storage \cap pnt : substance) \end{aligned}$$

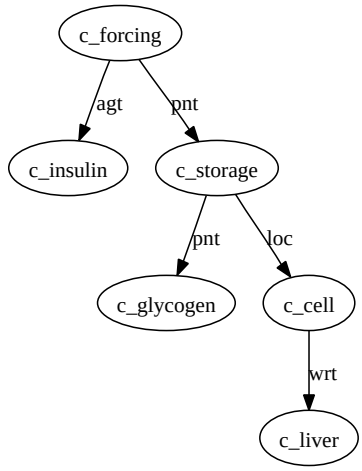
How would the search engine identify the sentence of interest as the answer to the query? It is worth observing that `forcing` is a subclass of `causing` and `glycogen` is a subclass of `substance`. One approach could use grammatical production rules and in a few steps derive the text concept from the query concept. This approach has been presented in Figure 4.3 on page 82.

Another procedure can use a graph matching algorithm. Ontoterms can be represented graphically in the form of direct acyclic tree graphs. In our case, we would need to match the query graph:



insulin[n] caused[v] storage[n] of[prep] substance[n]

against the article sentence graph:



insulin[n] forces[v] storage[n] of[prep] glycogen[n] in[prep] liver[n] cells[n]

Please observe that the underlying ontology with its class subsumption plays a vital role in such a matching, as we need to identify that `forcing` is a subclass of `causing` and `glycogen` is a subclass of `substance` for the graphs to match.

The graph matching has the advantage of respecting the commutativity of set intersection. Additionally it can very easily deal with inverse relations by simply honoring the direction of the arc in the graph.

The need of comparing the query graph to all graphs in the database can be easily eliminated by indexing the database graphs by the primitive classes appearing in them, which are conceived in the current context as graph nodes. The query semantics would only need to be matched against those graphs, which contain all the concepts appearing in the query according to the index.

Even though the core of the engine would be relatively easy to implement, the amount of work would be substantial due to the need of downloading many articles, setting up a database for them, indexing them, etc. However, the SIABO project is currently working on an ontological search engine, which works very much along similar lines.

9.3.1 Ad–hoc concepts

A keyword–based search has a certain advantage over an ontological semantic search. If a given word does not correspond to any class in the ontology, the

keyword search will still be able to perform some search involving such a word. It is possible to add such a functionality to the ontological search engine quite easily.

This can be achieved by introducing ad–hoc concepts. If a word does not correspond to any class in the ontology, we can generate a new class on the fly. Such a class would be identified by the given word. Obviously, we would not know where this class should be placed in the ontology. We may simply put it under the top concept. The ad–hoc class created in such a way allows the ontological search engine to work as if it was a keyword, i.e. only exact match will be considered as a response to the query. While this approach is relatively primitive, it allows at least some search results to be returned in case that the ontology does not contain what was queried about.

Chapter 10

Conclusion

In Chapter 1 we have outlined the motivation behind this work, including the fact that we would like to offer search–friendly semantics that surpasses the capabilities of the keyword–based search. That should be achieved by merging formal domain ontologies with natural language semantics.

In Chapter 2 we have presented formalisms that are useful for representing either formal ontologies, natural language semantics, or both. We also gave examples about how they can be utilized. First Order Predicate Logic has been used throughout the thesis as a common metalogical framework language, in which other ideas and formalisms have been presented. This has functioned as a glue between the logical methods presented and the computation procedures developed in later chapters, where we used logic programming.

Chapter 3 provided a general introduction to ontologies, with focus on formal ones. In this chapter we have not yet dwelved into natural language semantics. Instead, we have presented how to use the logical framework of formal ontologies to extract non–trivial information from them with the help of logic programming.

In Chapter 4 we explored the notion of types. We demonstrated how much variety exists between types as used in logic, ontologies, and programming languages. Last but not least we have presented how generative ontologies work to provide the infinitude of ontological types, or "ontoterms".

In Chapter 5 we had a closer look at natural language – its grammar, meaning, and some problems related to extracting that meaning. We have presented some ways of how to merge the formal ontologies with natural language semantics.

In Chapter 6 we looked at the state–of–the–art in ontological semantics and we have compared it to our approach.

In Chapter 7 we have answered the question of how to approach the problem of relating computationally the ontological semantics and a natural language fragment. We have discussed how to use practical logic programming tools, such as Prolog, for that purpuse, while keeping as close to the underlying logical specification as possible. We have elaborated several approaches, each with its own advantages, and described them in detail.

In Chapter 8 we went one step further and try to use similar methods as in Chapter 7 for dealing with unrestricted natural language. However, we have moved further away from the pure logical specification in order to be able to deal with unrestricted versions of natural language. The approach presented in this chapter introduces a novelty, i.e. the utilization of the ontological meaning earlier in the process understanding of the language than the grammar itself.

We have presented some final thoughts and the discussion of further work in Chapter 9.

Appendix A

Investigations of the conjunctions' reading (distributive vs. collective) of Wikipedia Insulin sentences

The following is an investigation of the reading of conjunctions in a scientific text, i.e. the Wikipedia entry on Insulin.

For each sentence:

- The conjunction has been identified with a frame
- The reading has been distinguished (whether it is collective or distributive)
- The syntactic level of the conjunction has been identified (NP level, sentence level, etc.)
- The relevance of the sentence has been decided (some sentences contributing a lot to the given scientific text, and some not being interesting from that point of view)

A summary follows, where the findings of this study are presented.

A.1 Sample sentences

1: Insulin is a hormone with intensive effects on both metabolism and several other body systems.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

2: Insulin causes most of the body's cells to take up glucose from the blood, storing it as glycogen in the liver and muscle, and stops use of fat as an energy source.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

3: Insulin causes most of the body's cells to take up glucose from the blood, storing it as glycogen in the liver and muscle, and stops use of fat as an energy source.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

4: When insulin is absent, glucose is not taken up by most body cells and the body begins to use fat as an energy source.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

5: Insulin is a peptide hormone composed of 51 amino acid residues and has a molecular weight of 5808 Da.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

6: Bovine insulin differs from human in only three amino acid residues, and porcine insulin in one.

The reading is distributive. Ellipsis. Sentence is irrelevant.

7: Insulin in some invertebrates is quite close to human insulin, has similar effects inside cells, and is produced very similarly.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

8: Insulin is produced in the pancreas, and released when any of several stimuli are detected.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

9: These include protein ingestion, and glucose in the blood.

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

10: In target cells, they initiate a signal transduction which has the effect of increasing glucose uptake [and] storage.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

11: Production [and] secretion are largely independent; prepared insulin is stored awaiting secretion.

The reading is collective. Conjunction at NP level. Relevance is questionable.

12: Both C-peptide [and] mature insulin are biologically active.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

13: Cell components [and] proteins in this image are not to scale.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

14: These modifications of proinsulin remove the center portion of the molecule, from the C- [and] N- terminal ends of proinsulin.

Conjunction at NP level. Relevance is questionable.

15: The remaining polypeptides, the B- [and] A- chains, are bound together by disulfide bonds/disulphide bonds.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

16: Confusingly, the primary sequence of proinsulin goes in the order "B-C-A", since B [and] A chains were identified on the basis of mass, and the C peptide was discovered after the others.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

17: Confusingly, the primary sequence of proinsulin goes in the order "B-C-A", since B and A chains were identified on the basis of mass, [and] the C peptide was discovered after the others.

The reading is distributive. Conjunction at S level. Sentence is relevant.

18: Glucose goes into the glycolysis [and] the respiratory cycle where multiple high-energy ATP molecules are produced by oxidation

The reading is distributive. Conjunction at NP level. Sentence is relevant.

19: Dependent on ATP levels, [and] hence blood glucose levels, the ATP-controlled potassium channels close and the cell membrane depolarizes

The reading is distributive. Conjunction at NP level. Sentence is relevant.

20: Dependent on ATP levels, and hence blood glucose levels, the ATP-controlled potassium channels close [and] the cell membrane depolarizes

The reading is distributive. Conjunction at S level. Sentence is relevant.

21: On depolarization, voltage controlled calcium channels open [and] calcium flows into the cells

The reading is distributive. Conjunction at S level. Sentence is relevant.

22: An increased calcium level causes activation of phospholipase C, which cleaves the membrane phospholipid phosphatidyl inositol 4,5-bisphosphate into inositol 1,4,5-triphosphate [and] diacylglycerol.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

23: This allows the release of Ca^{2+} from the ER via IP3 gated channels, [and] further raises the cell concentration of calcium.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

24: This is the main mechanism for release of insulin [and] regulation of insulin synthesis.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

25: In addition some insulin synthesis [and] release takes place generally at food intake, not just glucose or carbohydrate intake, and the beta cells are also somewhat influenced by the autonomic nervous system.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

26: In addition some insulin synthesis and release takes place generally at food intake, not just glucose or carbohydrate intake, [and] the beta cells are also somewhat influenced by the autonomic nervous system.

The reading is distributive. Conjunction at S level. Sentence is relevant.

27: Other substances known to stimulate insulin release include amino acids from ingested proteins, acetylcholine, released from vagus nerve endings, cholecystokinin, released by enteroendocrine cells of intestinal mucosa and glucose-dependent insulinotropic peptide.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

28: This is thought to avoid downregulation of insulin receptors in target cells and to assist the liver in extracting insulin from the blood.

The reading is distributive. Sentence is relevant.

29: Activation of insulin receptors leads to internal cellular mechanisms that directly affect glucose uptake by regulating the number and operation of protein molecules in the cell membrane that transport glucose into the cell.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

30: The genes that specify the proteins that make up the insulin receptor in cell membranes have been identified and the structure of the interior, cell membrane section, and now, finally after more than a decade, the extra-membrane structure of receptor.

Ellipsis. Relevance is questionable.

31: Two types of tissues are most strongly influenced by insulin, as far as the stimulation of glucose uptake is concerned: muscle cells and fat cells.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

32: The former are important because of their central role in movement, breathing, circulation, etc, and the latter because they accumulate excess food energy against future needs.

Ellipsis. Relevance is questionable.

33: Effect of insulin on glucose uptake and metabolism.

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

34: These include: translocation of Glut-4 transporter to the plasma membrane [and] influx of glucose, glycogen synthesis, glycolysis and fatty acid synthesis.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

35: These include: translocation of Glut-4 transporter to the plasma membrane and influx of glucose, glycogen synthesis, glycolysis [and] fatty acid synthesis.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

36: Control of cellular intake of certain substances, most prominently glucose in muscle [and] adipose tissue.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

37: Increase of DNA replication [and] protein synthesis via control of amino acid uptake.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

38: Increased glycogen synthesis - insulin forces storage of glucose in liver cells in the form of glycogen; lowered levels of insulin cause liver cells to convert glycogen to glucose [and] excrete it into the blood.

The reading is distributive. Relevance is questionable.

39: Decreased gluconeogenesis - decreases production of glucose from non-sugar substrates, primarily in the liver ; lack of insulin causes glucose production from assorted substrates in the liver [and] elsewhere.

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

40: Once an insulin molecule has docked onto the receptor [and] effected its action, it may be released back into the extracellular environment or it may be degraded by the cell.

The reading is collective. Relevance is questionable.

41: They do not require insulin to absorb glucose, unlike muscle [and] adipose tissue, and they have very small internal stores of glycogen.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

42: They do not require insulin to absorb glucose, unlike muscle and adipose tissue, [and] they have very small internal stores of glycogen.

The reading is distributive. Conjunction at S level. Sentence is relevant.

43: Glycogen stored in liver cells can be converted to glucose, [and] released into the blood, when glucose from digestion is low or absent, and the glycerol backbone in triglycerides can also be used to produce blood glucose.

The reading is distributive. Relevance is questionable.

44: Glycogen stored in liver cells can be converted to glucose, and released into the blood, when glucose from digestion is low or absent, [and] the glycerol backbone in triglycerides can also be used to produce blood glucose.

The reading is distributive. Conjunction at S level. Sentence is relevant.

45: Sufficient lack of glucose [and] scarcity of these sources of glucose can dramatically make itself manifest in the impaired functioning of the central nervous system; dizziness, speech problems, and even loss of consciousness, can occur.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

46: Sufficient lack of glucose and scarcity of these sources of glucose can dramatically make itself manifest in the impaired functioning of the central nervous system; dizziness, speech problems, [and] even loss of consciousness, can occur.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

47: Endogenous causes of insulin excess are very rare, [and] the overwhelming majority of insulin-excess induced hypoglycemia cases are iatrogenic and usually accidental.

The reading is distributive. Conjunction at S level. Sentence is relevant.

48: Endogenous causes of insulin excess are very rare, and the overwhelming majority of insulin-excess induced hypoglycemia cases are iatrogenic [and] usually accidental.

Sentence is relevant. Sentence is irrelevant.

49: Metabolic syndrome - a poorly understood condition first called Syndrome X by Gerald Reaven, Reaven's Syndrome after Reaven, CHAOS in Australia, [and] sometimes prediabetes.

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

50: It is characterized by elevated blood pressure, dyslipidemia, [and] increased waist circumference.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

51: Commonly, morbidities such as essential hypertension, obesity, Type 2 diabetes, [and] cardiovascular disease develop.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

52: Polycystic ovary syndrome - a complex syndrome in women in the reproductive years where there is anovulation [and] androgen excess commonly displayed as hirsutism.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

53: The host cells are then allowed to grow [and] reproduce normally, and due to the inserted human DNA, they produce a synthetic version of human insulin.

The reading is distributive. Relevance is questionable.

54: The host cells are then allowed to grow and reproduce normally, [and] due to the inserted human DNA, they produce a synthetic version of human insulin.

Relevance is questionable.

55: According to a survey that the International Diabetes Federation conducted in 2002 on the access to [and] availability of insulin in its member countries, approximately 70% of the insulin that is currently sold in the world is recombinant, biosynthetic 'human' insulin.

The reading is distributive. Ellipsis. Relevance is questionable.

56: Also, the International Diabetes Federation's position statement is very clear in stating that "there is NO overwhelming evidence to prefer one species

of insulin over another" and "[modern, highly-purified] animal insulins remain a perfectly acceptable alternative.

Sentence is irrelevant.

57: Selecting the 'right' dose and timing.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

58: Adjusting dosage and timing to fit food intake timing, amounts, and types.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

59: Adjusting dosage and timing to fit exercise undertaken.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

60: Adjusting dosage, type, and timing to fit other conditions, for instance the increased stress of illness.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

61: The dosage is non-physiological in that a subcutaneous bolus dose of insulin alone is administered instead of combination of insulin and C-peptide being released gradually and directly into the portal vein.

The reading is collective. Conjunction at NP level. Sentence is relevant.

62: These delay absorption of the insulin, adjust the pH of the solution to reduce reactions at the injection site, and so on.

Sentence is irrelevant.

63: They have absorption and activity characteristics not currently possible with subcutaneously injected insulin proper.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

64: Choosing insulin type and dosage/timing should be done by an experienced medical professional working closely with the diabetic patient.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

65: these begin to work within 5 to 15 minutes [and] are active for 3 to 4 hours.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

66: Short-acting, such as "regular" insulin starts working within 30 minutes [and] is active about 5 to 8 hours.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

67: Intermediate-acting, such as "NPH", or "lente" insulin starts working in 1 to 3 hours [and] is active 16 to 24 hours.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

68: Long-acting, such as "ultralente" insulin starts working in 4 to 6 hours, [and] is active 24 to 28 hours.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

69: "Insulin glargine" [and] "Insulin detemir" both insulin analogs which start working within 1 to 2 hours and continue to be active, without major peaks or dips, for about 24 hours, although this varies in many individuals.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

70: "Insulin glargine" and "Insulin detemir" both insulin analogs which start working within 1 to 2 hours [and] continue to be active, without major peaks or dips, for about 24 hours, although this varies in many individuals.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

71: A mixture of NPH [and] regular insulin - starts working in 30 minutes and is active 16 to 24 hours.

The reading is collective. Conjunction at NP level. Sentence is relevant.

72: A mixture of NPH and regular insulin - starts working in 30 minutes and is active 16 to 24 hours.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

73: 25 in India claiming it eliminates the risk of contracting diseases such as BSE and CJD associated with insulin derived from pigs and cattle.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

74: 25 in India claiming it eliminates the risk of contracting diseases such as BSE and CJD associated with insulin derived from pigs and cattle.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

75: Hence, both a long-acting insulin and a short-acting insulin are typically used.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

76: Advantages to the patient are better control over background or 'basal' insulin dosage, bolus doses calculated to fractions of a unit, and calculators in the pump that may help with determining 'bolus' infusion dosages.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

77: The limitations are cost, the potential for hypoglycemic and hyperglycemic episodes, catheter problems, and no "closed loop" means of controlling insulin delivery based on current blood glucose levels.

The reading is distributive. Conjunction at ADJ level. Sentence is relevant.

78: The limitations are cost, the potential for hypoglycemic and hyperglycemic episodes, catheter problems, and no "closed loop" means of controlling insulin delivery based on current blood glucose levels.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

79: In addition, indwelling catheters pose the risk of infection and ulceration, and some patients may also develop lipodystrophy due to the infusion

sets.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

80: In addition, indwelling catheters pose the risk of infection and ulceration, and some patients may also develop lipodystrophy due to the infusion sets.

The reading is distributive. Conjunction at S level. Sentence is relevant.

81: Insulin pumps require care and effort to use correctly.

The reading is collective. Conjunction at NP level. Relevance is questionable.

82: Food and Drug Administration approved the use of Exubera, the first inhalable insulin.

Conjunction at name level. Relevance is questionable.

83: Inhaled insulin has similar efficacy to injected insulin, both in terms of controlling glucose levels and blood half-life.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

84: Currently, inhaled insulin is short acting and is typically taken before meals; an injection of long-acting insulin at night is often still required.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

85: Following its commercial launch in 2005 in the UK, it was not recommended by National Institute for Health and Clinical Excellence for routine use, except in cases where there is ""proven injection phobia diagnosed by a psychiatrist or psychologist"".

Conjunction at name level. Relevance is questionable.

86: Similarly, Eli Lilly and Company ended its efforts to develop its Air inhaled insulin in March 2008.

Conjunction at name level. Relevance is questionable.

87: Jet injection had different insulin delivery peaks and durations as compared to needle injection.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

88: Both electricity using iontophoresis [and] ultrasound have been found to make the skin temporarily porous.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

89: Researchers have produced a watch-like device that tests for blood glucose levels through the skin [and] administers corrective doses of insulin through pores in the skin.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

90: However, insulin is a protein, which are digested in the stomach [and] gut and in order to be effective at controlling blood sugar, can not be taken orally.

The reading can be collective or distributive. Conjunction at NP level. Sentence is relevant.

91: However, insulin is a protein, which are digested in the stomach and gut [and] in order to be effective at controlling blood sugar, can not be taken orally.

The reading is distributive. Conjunction at S level. Sentence is relevant.

92: In a Phase I study, VIAtab delivered insulin to the blood stream quickly [and] resembled the first-phase insulin release spike found in healthy individuals.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

93: The company claims that an oral insulin therapy would be more convenient than currently available injectable or inhalable therapies, [and] they expect that convenience to result in increased insulin usage among the currently underserved early-stage patients with Type 2 diabetes, thus helping to create better long-term outcomes for that patient population.

The reading is distributive. Conjunction at S level. Sentence is relevant.

94: Transplantation of an entire pancreas is difficult [and] relatively uncommon.

The reading is distributive. Conjunction at ADJ level. Sentence is irrelevant.

95: However, researchers at the University of Illinois at Chicago have slightly modified the Edmonton Protocol procedure for islet cell transplantation [and]

achieved insulin independence in diabetes patients with fewer but better-functioning pancreatic islet cells.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

96: The central problem for those requiring external insulin is picking the right dose of insulin and the right timing.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

97: The increased insulin level causes glucose absorption and storage in cells, reduces glycogen to glucose conversion, reducing blood glucose levels, and so reducing insulin release.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

98: The increased insulin level causes glucose absorption and storage in cells, reduces glycogen to glucose conversion, reducing blood glucose levels, and so reducing insulin release.

Relevance is questionable.

99: The result is that the blood glucose level rises somewhat after eating, and within an hour or so, returns to the normal 'fasting' level.

Relevance is questionable.

100: In additions, fats and proteins cause delays in absorption of glucose from carbohydrate eaten at the same time.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

101: Because of the complex and interacting factors, it is, in principle, impossible to know for certain how much insulin is needed to 'cover' a particular meal to achieve a reasonable blood glucose level within an hour or two after eating.

The reading can be collective or distributive. Conjunction at ADJ level. Relevance is questionable.

102: Non-diabetics' beta cells routinely and automatically manage this by continual glucose level monitoring and insulin release.

The reading is distributive. Conjunction at ADJ level. Relevance is questionable.

103: Non-diabetics' beta cells routinely and automatically manage this by continual glucose level monitoring and insulin release.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

104: All such decisions by a diabetic must be based on experience and training and, further, specifically based on the individual experience of the patient.

The reading is collective. Conjunction at NP level. Relevance is questionable.

105: But it is not straightforward and should never be done by habit or routine.

The reading is distributive. Conjunction at VP level. Sentence is relevant.

106: For example, some patients with diabetes require more insulin after drinking skim milk than they do after taking an equivalent amount of fat, protein, carbohydrate, and fluid in some other form.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

107: Maintaining the basal rate and the bolus rate is a continuous balancing act that people with insulin-dependent diabetes must manage each day.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

108: NPH/isophane, lente, ultralente, glargine, and detemir may be used for this purpose.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

109: The advantage of NPH is its low cost and the fact that you can mix it with short-acting forms of insulin, thereby minimizing the number of injections that must be administered.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

110: The disadvantage is that the activity of NPH is less steady [and] will peak 4-6 hours after administration, and this peak has the potential of causing hypoglycemia.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

111: NPH [and] regular insulin in combination are available as premixed solutions, which can sometimes simplify administration.

The reading is collective. Conjunction at NP level. Relevance is questionable.

112: The theoretical advantage of glargine [and] detemir is that they only need to be administered once a day, and they also have steady activity, generally without peaks, although in practice, many patients find that neither lasts a full 24 hours.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

113: The theoretical advantage of glargine and detemir is that they only need to be administered once a day, [and] they also have steady activity, generally without peaks, although in practice, many patients find that neither lasts a full 24 hours.

The reading is distributive. Conjunction at S level. Sentence is relevant.

114: Glargine [and] detemir are also signifincantly more expensive, and they cannot be mixed with other forms of insulin.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

115: Glargine and detemir are also signifincantly more expensive, [and] they cannot be mixed with other forms of insulin.

The reading is distributive. Conjunction at S level. Sentence is relevant.

116: Regular insulin, lispro, aspart [and] glulisine can be used for this purpose.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

117: Regular insulin should be given with about a 30 minute lead-time prior to the meal to be maximally effective [and] to minimize the possibility of hypoglycemia.

The reading is distributive. Sentence is relevant.

118: Lispro, aspart and glulisine are approved for dosage with the first bite of the meal, and may even be effective if given after completing the meal.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

119: Lispro, aspart and glulisine are approved for dosage with the first bite of the meal, and may even be effective if given after completing the meal.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

120: Insulin prescriptions generally specify fixed amounts of long-acting insulin to be given routinely, and fixed amounts of short-acting insulin prior to every meal.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

121: "Sample regimen using insulin NPH and regular insulin"

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

122: "Sample regimen using insulin glargine and insulin lispro"¡br /¿

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

123: A more complicated method that allows greater freedom with meal times and snacks is carb counting.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

124: The patient can use his or her total daily dose of insulin to estimate how many grams of carbohydrates will be "covered" by 1 unit of insulin, and using this result, the patient can estimate how many units of insulin should be administered depending on the carbohydrate concentration of their meal.

Sentence is irrelevant.

125: However, all dosages involve a fair degree of guesswork, and will seldom work consistently from one dosage to the next.

The reading is distributive. Conjunction at VP level. Relevance is questionable.

126: Poorly controlled diabetics are more prone than others to exhaustion and tiredness, and properly-administered insulin can relieve such symptoms.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

127: Poorly controlled diabetics are more prone than others to exhaustion and tiredness, and properly-administered insulin can relieve such symptoms.

The reading is distributive. Conjunction at S level. Sentence is relevant.

128: "Game of Shadows," by reporters Mark Fainaru-Wada and Lance Williams, includes allegations that Barry Bonds used insulin in the apparent belief that it would increase the effectiveness of the growth hormone he was taking.

The reading is collective. Conjunction at NP level. Sentence is irrelevant.

129: On top of this, non-prescribed insulin is a banned drug at the Olympics and other global competitions.

The reading is distributive. Conjunction at NP level. Sentence is irrelevant.

130: The use and abuse of exogenous insulin is claimed to be widespread amongst the bodybuilding community.

The reading is distributive. Conjunction at NP level. Relevance is questionable.

131: Insulin, human growth hormone and insulin-like growth factor 1 are self-administered by those looking to increase muscle mass beyond the scope offered by anabolic steroids alone.

The reading can be collective or distributive. Conjunction at NP level. Relevance is questionable.

132: This theory has been supported in recent years by top-level bodybuilders whose competition weight is in excess of of muscle, larger than that of competitors in the past, and with even lower levels of body fat.

Relevance is questionable.

133: The abuse of exogenous insulin carries with it an attendant risk of hypoglycemic coma [and] death when the amount used is in excess of that required to handle ingested carbohydrate.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

134: Acute risks include brain damage, paralysis, [and] death.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

135: Long-term risks may include development of type 2 diabetes, [and] potentially a lifetime dependency on exogenous insulin.

The reading is distributive. Conjunction at NP level. Sentence is relevant.

136: In addition, overly high levels of insulin are seen only in those with pathologies such as Type 2 diabetes mellitus, [and] only in some of those.

Sentence is irrelevant.

137: The typical person cannot have insulin overload [and] still have blood glucose levels which do not force symptoms of hypoglycemia.

The reading is collective. Relevance is questionable.

A.2 Summary

The reading can be collective or distributive: 15 16 22 45 87 90 101 103 131

The reading is collective: 11 40 61 71 81 104 111 128 137

The reading is distributive: 1 2 3 4 5 6 7 8 9 10 12 13 17 18 19 20 21 23 24 25 26 27 28 29 31 33 34 35 36 37 38 39 41 42 43 44 46 47 49 50 51 52 53 55 57 58 59 60 63 64 65 66 67 68 69 70 72 73 74 75 76 77 78 79 80 83 84 88 89 91 92 93 94 95 96 97 100 102 105 106 107 108 109 110 112 113 114 115 116 117 118 119 120 121 122 123 125 126 127 129 130 133 134 135

Conjunction at S level: 17 20 21 26 42 44 47 80 91 93 113 115 127

Conjunction at NP level: 1 2 9 10 11 12 13 14 15 16 18 19 22 24 25 27 29 31 33 34 35 36 37 39 41 45 46 49 50 51 52 57 58 59 60 61 63 64 69 71 73 74 75 76 78 79 81 83 87 88 90 96 97 100 103 104 106 107 108 109 111 112 114 116 118 120 121 122 123 126 128 129 130 131 133 134 135

Conjunction at ADJ level: 77 94 101 102

Conjunction at VP level: 3 4 5 7 8 23 65 66 67 68 70 72 84 89 92 95 105 110 119 125

Conjunction at name level: 82 85 86

Ellipsis: 6 30 32 55

Sentence is relevant: 1 2 3 4 5 7 8 10 12 13 17 18 19 20 21 23 24 25 26 27 28 29 36 37 41 42 44 47 48 50 51 61 71 73 75 77 79 80 83 88 89 90 91 92 93 97 100 105 106 107 108 112 113 114 115 116 117 126 127 133 134 135

Sentence is irrelevant: 6 9 33 39 48 49 56 62 94 121 122 124 128 129 136

Relevance is questionable: 11 14 15 16 22 30 31 32 34 35 38 40 43 45 46 52 53 54 55 57 58 59 60 63 64 65 66 67 68 69 70 72 74 76 78 81 82 84 85 86 87 95 96 98 99 101 102 103 104 109 110 111 118 119 120 123 125 130 131 132 137

Appendix B

Investigations of the type of relative clauses of the Wikipedia Insulin article sentences

The following is an investigation of the type of relative clauses occuring in a scientific text, i.e. the Wikipedia entry on Insulin.

Insulin text had 172 sentences, of which: 13 had a "which" relative clause. 29 had a "that" relative clause. 1 had a "who" relative clause.

B.1 Which

These include protein ingestion, and glucose in the blood (from food which produces glucose when digested – characteristically this is carbohydrate, though not all types produce glucose and so an increase in blood glucose levels).

In target cells, they initiate a signal transduction which has the effect of increasing glucose uptake and storage.

One million to three million islets of Langerhans (pancreatic islets) form the endocrine part of the pancreas, which is primarily an exocrine gland.

An increased calcium level causes activation of phospholipase C, which cleaves the membrane phospholipid phosphatidyl inositol 4,5-bisphosphate into inositol

1,4,5-triphosphate and diacylglycerol.

Metabolic syndrome a poorly understood condition first called Syndrome X by Gerald Reaven, Reaven's Syndrome after Reaven, CHAOS in Australia (from the signs which seem to travel together), and sometimes prediabetes.

Slight variations of the human insulin molecule are called insulin analogs, (technically "insulin receptor ligands") so named because they are not technically insulin, rather they are analogs which retain the hormone's glucose management functionality.

"Insulin glargine" and "Insulin detemir" both insulin analogs which start working within 1 to 2 hours and continue to be active, without major peaks or dips, for about 24 hours, although this varies in many individuals.

However, insulin is a protein, which are digested in the stomach and gut and in order to be effective at controlling blood sugar, can not be taken orally.

Because of the complex and interacting factors, it is, in principle, impossible to know for certain how much insulin (and which type) is needed to 'cover' a particular meal to achieve a reasonable blood glucose level within an hour or two after eating.

NPH and regular insulin in combination are available as premixed solutions, which can sometimes simplify administration.

The typical person cannot have insulin overload and still have blood glucose levels which do not force symptoms of hypoglycemia.

Nicolae Paulescu, a professor of physiology at the University of Medicine and Pharmacy in Bucharest was the first one to isolate insulin, which he called at that time pancrein, and published his work in 1921 that had been carried out in Bucharest.

Banting suggested that they try to use fetal calf pancreas, which had not yet developed digestive glands; he was relieved to find that this method worked well.

B.2 That

Activation of insulin receptors leads to internal cellular mechanisms that directly affect glucose uptake by regulating the number and operation of protein molecules in the cell membrane that transport glucose into the cell.

The genes that specify the proteins that make up the insulin receptor in cell membranes have been identified and the structure of the interior, cell membrane section, and now, finally after more than a decade, the extra-membrane structure of receptor (Australian researchers announced the work 2Q 2006).

It is a protein that has been highly conserved across evolutionary time.

According to a survey that the International Diabetes Federation conducted in 2002 on the access to and availability of insulin in its member countries, approximately 70

Also, the International Diabetes Federation's position statement is very clear in stating that "there is NO overwhelming evidence to prefer one species of insulin over another" and "[modern, highly-purified] animal insulins remain a perfectly acceptable alternative.

The dosage is non-physiological in that a subcutaneous bolus dose of insulin alone is administered instead of combination of insulin and C-peptide being released gradually and directly into the portal vein.

Advantages to the patient are better control over background or 'basal' insulin dosage, bolus doses calculated to fractions of a unit, and calculators in the pump that may help with determining 'bolus' infusion dosages.

Researchers have produced a watch-like device that tests for blood glucose levels through the skin and administers corrective doses of insulin through pores in the skin.

The company claims that an oral insulin therapy would be more convenient than currently available injectable or inhalable therapies, and they expect that convenience to result in increased insulin usage among the currently underserved early-stage patients with Type 2 diabetes, thus helping to create better long-term outcomes for that patient population.

The result is that the blood glucose level rises somewhat after eating, and within an hour or so, returns to the normal 'fasting' level.

Maintaining the basal rate and the bolus rate is a continuous balancing act that people with insulin-dependent diabetes must manage each day.

The advantage of NPH is its low cost and the fact that you can mix it with short-acting forms of insulin, thereby minimizing the number of injections that must be administered.

The disadvantage is that the activity of NPH is less steady and will peak 4-6 hours after administration, and this peak has the potential of causing hypoglycemia.

The theoretical advantage of glargine and detemir is that they only need to be administered once a day, and they also have steady activity, generally without peaks, although in practice, many patients find that neither lasts a full 24 hours.

A more complicated method that allows greater freedom with meal times and snacks is "carb counting.

"Game of Shadows," by reporters Mark Fainaru-Wada and Lance Williams, includes allegations that Barry Bonds used insulin in the apparent belief that it would increase the effectiveness of the growth hormone he was (also alleged to be) taking.

Their rationale is that since insulin and HGH act synergistically to promote growth, and since IGF-1 is a primary mediator of musculoskeletal growth, the 'stacking' of insulin, HGH and IGF-1 should offer a synergistic growth effect on skeletal muscle.

This theory has been supported in recent years by top-level bodybuilders whose competition weight is in excess of of muscle, larger than that of competitors in the past, and with even lower levels of body fat.

The abuse of exogenous insulin carries with it an attendant risk of hypoglycemic coma and death when the amount used is in excess of that required to handle ingested carbohydrate.

In the non-diabetic, the feedback control mechanism connecting insulin release and blood glucose level is very effective, and it is not possible to adjust it except that blood glucose levels rise slightly during digestion and absorption of glucose.

The decrease in blood glucose levels is directly attributable to release of insulin, and that release ceases as blood glucose levels drop.

On testing the urine they found that there was sugar in the dog's urine, establishing for the first time a relationship between the pancreas and diabetes.

Nicolae Paulescu, a professor of physiology at the University of Medicine and Pharmacy in Bucharest was the first one to isolate insulin, which he called at that time pancrein, and published his work in 1921 that had been carried out in Bucharest.

In October 1920 Canadian Frederick Banting was reading one of Minkowski's papers and concluded that it is the very digestive secretions that Minkowski had originally studied that were breaking down the islet secretion(s), thereby making it impossible to extract successfully.

Several weeks later it was clear the second run was also a success, and he helped publish their results privately in Toronto that November.

Banting suggested that they try to use fetal calf pancreas, which had not yet developed digestive glands; he was relieved to find that this method worked well.

However, the extract was so impure that Thompson suffered a severe allergic reaction, and further injections were canceled.

Banting, insulted that Best was not mentioned, shared his prize with Best, and Macleod immediately shared his with Collip.

Prior to that, insulin was sold in different strengths, including U-80 (80 units per milliliter) and U-40 formulations (40 units per milliliter), so the effort to "standardize" the potency aimed to reduce dosage errors and ease doctors' job of prescribing insulin for patients.

B.3 Who

Macleod, who was not entirely impressed with his idea so many before him had tried and failed.

Appendix C

Mercury implementation of the ontological semantics in categorial grammar

Below we present full implementation of the ideas from Section 5.12 on page 98.

```
:-module a.

:-interface.

:-import_module io.
:-pred main(io::di,io::uo) is det.

:-implementation.

%% Simple concept in skeleton ontology:
:-type alpha --->
        child;
        tall;
        physical;
        action;
        person;
        running;
```

```
        smiling;
        animate;
        vitamin;
        vitamin_c;
        acid;
        i(alpha,alpha);
        p(role,alpha).

:-pred sub(alpha::in,alpha::out) is nondet.

sub(child,person).
sub(person,physical).
sub(person,animate).
sub(running,action).
sub(smiling,action).
sub(vitamin_c,vitamin).


:-type role --->
    tmp % temporal aspects (generic role)
  ; loc % location, position
  ; prp % purpose, function
  ; wrt % with respect to
  ; chr % characteristic (property ascription)
  ; cum % cum (i.e., with accompanying)
  ; bmo % by means of, instrument, via
  ; cby % caused by
  ; cau % causes
  ; cmp % comprising, has part
  ; pof % part of
  ; agt % agent of act or process
  ; pnt % patient of act or process
  ; src % source of act or process
  ; rst % result of act or process
  ; dst % destination of moving process
  .

:-type item --->
        b(pred(item,item));
```

```
        f(pred(item,item));
        e(alpha).

:-inst item_inst --->
        b(pred(in(item_inst),out(item_inst)) is semidet);
        f(pred(in(item_inst),out(item_inst)) is semidet);
        e(ground).

:-type llist --->
        n;
        l(item,llist).

:-inst llist_inst --->
        n;
        l(item_inst,llist_inst).

:-import_module solutions.

:-pred isa(alpha::in,alpha::in) is semidet.

isa(C,C).
isa(C,A):-sub(C,P),isa(P,A).
isa(i(X,_Y),Z):-isa(X,Z).
isa(i(_X,Y),Z):-isa(Y,Z).


:-type word --->
        w_the;w_a;w_kid;w_tall;w_runs;w_smiles;
        w_vitamin;w_c;w_ascorbic;w_acid.

%%Lexical entries.
:-pred lex(word::in,item::out(item_inst)) is det.

:-pred p_tall(item::in,item::out(item_inst)) is semidet.
:-pred p_the(item::in,item::out(item_inst)) is semidet.
:-pred p_runs(item::in,item::out(item_inst)) is semidet.
:-pred p_smiles(item::in,item::out(item_inst)) is semidet.
:-pred p_c(item::in,item::out(item_inst)) is semidet.
:-pred p_ascorbic(item::in,item::out(item_inst)) is semidet.
```

```
p_tall(e(C),e(i(tall,C))):-isa(C,physical).
p_runs(e(C),e(i(running,p(agt,C)))):-isa(C,animate).
p_smiles(e(C),e(i(smiling,p(agt,C)))):-isa(C,animate).
p_the(e(I),e(I)).
p_c(e(C),e(vitamin_c)):-isa(C,vitamin).
p_ascorbic(e(C),e(vitamin_c)):-isa(C,acid).

lex(w_kid,e(child)).
lex(w_tall,f(p_tall)).
lex(w_runs,b(p_runs)).
lex(w_smiles,b(p_smiles)).
lex(w_the,f(p_the)).
lex(w_a,f(p_the)).
lex(w_vitamin,e(vitamin)).
lex(w_c,b(p_c)).
lex(w_acid,e(acid)).
lex(w_ascorbic,f(p_ascorbic)).

:-pred r(llist::in(llist_inst),item::out(item_inst)) is nondet.

:-pred a(llist,
         llist,
         llist).

:-mode a(out(llist_inst),
         out(llist_inst),
         in(llist_inst)) is nondet.

a(l(X,n),l(Y,T),l(X,l(Y,T))).
a(l(H,L),M,l(H,T)):-a(L,M,T).

r(l(X,n),X).

r(G,E1):-a(G1,G2,G),r(G1,f(P)),r(G2,E2),P(E2,E1).

r(G,E1):-a(G1,G2,G),r(G1,E2),r(G2,b(P)),P(E2,E1).
```

```
:-pred m(text::in,alpha::out) is nondet.

m(T,C):-
        map_lex(T,L),
        r(L,e(C)).

:-pred work(alpha::out) is nondet.

work(C):-
        m(t(w_the,t(w_tall,t(w_kid,t(w_smiles,tnil)))),C).
        %m(t(w_the,t(w_tall,t(w_vitamin,t(w_smiles,tnil)))),C).

:-type text --->
        tnil;
        t(word,text).

:-pred map_lex(text::in,llist::out(llist_inst)) is nondet.

map_lex(tnil,n).
map_lex(t(W,T),l(I,MT)):-lex(W,I),map_lex(T,MT).

main(!IO) :-
        solutions(work,Cs),
        io.write(Cs,!IO),
        io.nl(!IO).
```

Appendix D

SML implementation of the Earley parser

Earley parser is an efficient polynomial time parser, with n^3 time complexity [4] (see also [25]). Below we present the implementation of it in the Standard Meta Language.

D.1 Signature

```
signature Earley =
sig

    (* Types of terminals and nonterminals*)

    type t = char
    type n = string

    (* Terminals and nonterminals are symbols *)

    type symbol

    val t : t -> symbol
    val n : n -> symbol
```

```
    (* [recognize rules root sequence] will return true if
    sequence belongs to language described by rules and
    root. rules is the list of grammar rules and root is the
    starting symbol.*)

    val recognize : (n*symbol list) list -> n -> t list -> bool

end
```

D.2 Implementation

```
structure Earley :> Earley =
struct

(* User will use this simple way of rules specification *)

type t = char
type n = string

(* And below are internal types *)

(* We need to extend set of terminal symbols with terminator *)
datatype terminal
  = is of t (* any input symbol *)
  | terminator (* special terminator symbol *)

datatype symbol
  = nter of n (* nonterminal, name of the syntactic class *)
  | ter of terminal (* terminal symbol *)


fun t t = ter (is t)
fun n n = nter n

datatype lhsSymbol
  = lhs of n (* nonterminal, name of the syntactic class *)
```

```
  | phi (* special symbol used in starting set *)

type rules
  = (n, symbol list list) Polyhash.hash_table

exception notFound

datatype state = state of
         lhsSymbol (* lhs of the rule*)
         * (symbol list) (* already parsed symbols *)
         * (symbol list) (* not yet parsed symbols *)
         * terminal (* one look-ahead symbol *)
         * int (* where parsing started *)

fun makeStartState n
  = state (phi,
           [],
           [nter n],
           terminator,
           0)

(* type input = terminal Array.array *)

fun input ts
  = Array.fromList ((List.map is ts)@[terminator])

(* We represent set as map from set elements to unit *)
type stateSet = (state,unit) Polyhash.hash_table

fun createEmptyStateSet () =
    let
        val emptyStateSet : stateSet
          = Polyhash.mkPolyTable (10,notFound)
    in
        emptyStateSet
    end

exception internalError
```

```
fun createStateSetArray size startState =
    let
        val a = Array.array (size,createEmptyStateSet ())
    in
        if size < 1
        then raise internalError
        else (Array.update a 0 startState ; a)
    end

fun predictor (nonterminal,lookAhead,rules,started) =
    let
        fun aux (nonterminal,[],lookAhead,started)
          = []
          | aux (nonterminal,rhs::rhss,lookAhead,started)
            = state (lhs nonterminal,[],rhs,lookAhead,started)
              :: aux (nonterminal,rhss,lookAhead,started)
        val alternatives = Polyhash.find rules nonterminal
    in
        aux (nonterminal,alternatives,lookAhead,started)
    end

(* isSubstring (l,a,p) checks if list l is a substring of
array a starting at position p of the array.*)

(* fun isSubstring ([],_,_) = true
  | isSubstring (terminator::rest,input,position)
    = position >= Array.length input
  | isSubstring ((l terminal)::rest,input,position)
    = Array.sub (input,position) = terminal
      andalso isSubstring (rest,input,position+1)*)

fun completer ([],_) = []
  | completer ((state (lhs1,
                       beforeDot,
                       (nter nonterminal)::afterDot,
                       lookAhead,
                       started))::states,
               lhs2)
    = if nonterminal = lhs2
```

```
      then
          (state (lhs1,
                  beforeDot@[nter nonterminal],
                  afterDot,
                  lookAhead,
                  started))::(completer (states,lhs2))
      else
          completer (states,lhs2)
  | completer ((state (_,
                       _,
                       (ter terminal)::_,
                       _,
                       _))::states,
               lhs2)
    = completer (states,lhs2)
  | completer ((state (_,
                       _,
                       [],
                       _,
                       _))::states,
               lhs2)
    = completer (states,lhs2)

(* expand returns pair of lists, where first list contains
states for this position, second for the next position. *)

exception whatToDo (* Get rid of this TODO*)

fun expand (position,
            state (_,
                   _,
                   (nter nonterminal)::(ter terminal)::_,
                   _,
                   _),
            _,
            rules,
            input)
  = (predictor(nonterminal,terminal,rules,position),[])

```

```
  | expand (position,
          state (_,
                 _,
                 (nter nonterminal1)::(nter nonterminal2)::_,
                 _,
                 _),
          _,
          rules,
          input)
    = raise whatToDo

  | expand (position,
          state (_,
                 _,
                 [nter nonterminal],
                 lookAhead,
                 _),
          _,
          rules,
          input)
    = (predictor(nonterminal,lookAhead,rules,position),[])

  | expand (position,
          state (left,
                 beforeDot,
                 (ter terminal)::afterDot,
                 lookAhead,
                 started),
          _,
          _,
          input)
    (* scanner *)
    = if Array.sub (input,position) = terminal
      then
          ([],[state(left,
                     beforeDot@[ter terminal],
                     afterDot,
                     lookAhead,
                     started)])
```

```
182       else
              ([],[])

    | expand (position,
              state (phi,
                     beforeDot,
187                  [],
                     lookAhead,
                     started),
              stateSetArray,
              rules,
192           input)
      = raise whatToDo

    | expand (position,
              state (lhs left,
197                  beforeDot,
                     [],
                     lookAhead,
                     started),
              stateSetArray,
202           rules,
              input)
      (* completer *)
      = if Array.sub (input,position) = lookAhead
        then
207         (completer (List.map (fn (a,b) => a) (Polyhash.listItems
                (Array.sub (stateSetArray,started))),
                      left),
             [])
        else
212         ([],[])

fun rules listRules =
    let
        val rulesMap : rules
          = Polyhash.mkPolyTable((length listRules) div 2,
217                              notFound)
        fun aux ((left,right)::listRules,rulesMap)
```

```
            = (case Polyhash.peek rulesMap left of
                    SOME rights => Polyhash.insert rulesMap (left,
                        right::rights)
                  | NONE => Polyhash.insert rulesMap (left,[right])
                ;
                aux (listRules,rulesMap))
            | aux ([],rulesMap) = rulesMap
    in
        aux (listRules,rulesMap)
    end

fun for (from,to,rulesMap,stateSetArray)
  = if to <= from

fun recognize rulesList root ts =
    let
        val rulesMap
          = rules rulesList
        val stateSetAray
          = createStateSetArray (length ts + 2) (makeStartState
              root)
        val input
          = input ts
        val numberOfStates
          = length ts + 2
    in
        for (0,numberOfStates,rulesMap,stateSetArray)
    end


(*****************************)
(**  TESTING                **)
(*****************************)

(* User probably wants to specify rules in this way: *)

val ae
  = [("E", [n "T"]),
     ("E", [n "E",t #"+",n "T"]),
```

```
257         ("T", [n "P"]),
            ("T", [n "T",t #"*",n "P"]),
            ("P", [t #"a"])
            ]

262 val res
      = recognize ae "E" (explode "a+a*a")

    end
```

Appendix E

SML implementation of the generative ontology and the lexicon

Below we present an implementation of the generative ontology and the lexicon written in the Standard Meta Language.

E.1 Signature

```
signature GenerativeOntology =
sig
  datatype simple_concept = simple of string

  val simple_concept_compare : simple_concept * simple_concept ->
      order

  datatype role = r of string

  val role_compare : role * role -> order

  datatype concept
    = cc of simple_concept * (role*concept) list
```

```
  (* Simple textual representation of a concept *)
  val c2s : concept -> string

  (* LaTeX tabular-based representation of a concept *)
  val c2latex : concept -> string

  (* TODO: Make isa faster. *)
  (* exception concept_not_found of string *)
  (* val isa : simple_concept -> simple_concept list *)

  datatype syntactic_category = syn of string

  val sc2s : syntactic_category -> string

  val syntactic_category_compare : syntactic_category *
      syntactic_category -> order

  datatype lexicon_item = li of syntactic_category * concept

  exception lex_not_found of string
  val lex : string -> lexicon_item list

  val aff : simple_concept * simple_concept -> role list
end
```

E.2 Implementation

```
structure GenerativeOntology :> GenerativeOntology =
struct

  fun poly_compare (x1,x2) =
    Int.compare (Polyhash.hash x1,Polyhash.hash x2)

  datatype simple_concept = simple of string

  fun sc2s (simple x) = x
```

```
fun simple_concept_compare (simple x,simple y) = String.compare
    (x,y)

datatype role = r of string

fun r2s (r x) = x

fun role_compare (r x,r y) = String.compare(x,y)

datatype concept
  = cc of simple_concept * (role*concept) list

fun p2latex (role,c) = "{\\it "^r2s role^"}:\\hspace{2 pt}"^
    c2latex c
and ps2latex [] = ""
  | ps2latex (p::ps) = "\\begin{math}\\left[\\begin{tabular}{@{}
      l@{}}"^p2latex p^foldr op^ "" (map (fn p=>"\\\\"^p2latex p)
      ps)^"\\end{tabular}\\right]\\end{math}"
and c2latex (cc(x,ps)) = sc2s x ^ ps2latex ps

fun p2s (role,c) = r2s role^":"^c2s c
and ps2s [] = ""
  | ps2s (p::ps) = "["^p2s p^foldr op^ "" (map (fn p=>","^p2s p)
      ps)^"]"
and c2s (cc(x,ps)) = sc2s x ^ ps2s ps

datatype syntactic_category = syn of string

fun sc2s (syn x) = x
fun syntactic_category_compare (syn x,syn y) = String.compare (x
    ,y)

datatype lexicon_item = li of syntactic_category * concept

exception lex_not_found of string

fun lex "," = [li(syn(","),cc(simple("nil"),[]))]
  | lex "3t3-l1" = [li(syn("N"),cc(simple("q_3t3l1"),[]))]
```

```
    | lex "abolish" = [li(syn("TV"),cc(simple("abolition"),[]))]
    | lex "activation" = [li(syn("N"),cc(simple("activation"),[]))
      ]
    | lex "activity" = [li(syn("N"),cc(simple("activity"),[]))]
    | lex "acts" = [li(syn("TV"),cc(simple("acting"),[]))]
    | lex "addition" = [li(syn("N"),cc(simple("addition"),[]))]
    | lex "adipocyte" = [li(syn("N"),cc(simple("adipocyte"),[]))]
    | lex "adipocytes" = [li(syn("N"),cc(simple("adipocyte"),[]))]
    | lex "age" = [li(syn("N"),cc(simple("age"),[]))]
    | lex "agents" = [li(syn("N"),cc(simple("agent"),[]))]
    | lex "allimin" = [li(syn("N"),cc(simple("allimin"),[]))]
    | lex "am" = [li(syn("IS"),cc(simple("is"),[]))]
    | lex "anti-fungal" = [li(syn("A"),cc(simple("antifungal"),[])
      )]
    | lex "are" = [li(syn("IS"),cc(simple("is"),[]))]
    | lex "auc" = [li(syn("N"),cc(simple("auc"),[]))]
    | lex "background" = [li(syn("N"),cc(simple("background"),[]))
      ]
    | lex "be" = [li(syn("BE"),cc(simple("nil"),[]))]
    | lex "betacell" = [li(syn("N"),cc(simple("betacell"),[]))]
    | lex "betacells" = [li(syn("N"),cc(simple("betacell"),[]))]
    | lex "binding" = [li(syn("N"),cc(simple("binding"),[])),li(
      syn("VBG"),cc(simple("binding"),[]))]
    | lex "blood" = [li(syn("N"),cc(simple("blood"),[]))]
    | lex "carrier" = [li(syn("N"),cc(simple("carrier"),[]))]
    | lex "caspase" = [li(syn("N"),cc(simple("caspase"),[]))]
    | lex "caspases" = [li(syn("N"),cc(simple("caspase"),[]))]
    | lex "causing" = [li(syn("VBG"),cc(simple("causing"),[]))]
    | lex "cell" = [li(syn("N"),cc(simple("cell"),[]))]
    | lex "cells" = [li(syn("N"),cc(simple("cell"),[]))]
    | lex "cellular" = [li(syn("A"),cc(simple("cellular"),[]))]
    | lex "central" = [li(syn("A"),cc(simple("central"),[]))]
    | lex "challenge" = [li(syn("N"),cc(simple("challenge"),[]))]
    | lex "characterization" = [li(syn("N"),cc(simple("
      characterization"),[]))]
    | lex "cloning" = [li(syn("N"),cc(simple("cloning"),[]))]
    | lex "combination" = [li(syn("N"),cc(simple("combination")
      ,[]))]
    | lex "compound" = [li(syn("N"),cc(simple("compound"),[]))]
```

```
    | lex "compounds" = [li(syn("N"),cc(simple("compound"),[]))]
    | lex "concentration" = [li(syn("N"),cc(simple("concentration"),[]))]
    | lex "concentrations" = [li(syn("N"),cc(simple("concentration"),[]))]
    | lex "controlled" = [li(syn("VBD"),cc(simple("control"),[])),li(syn("VBN"),cc(simple("control"),[]))]
    | lex "cpeptide" = [li(syn("N"),cc(simple("cpeptide"),[]))]
    | lex "death" = [li(syn("N"),cc(simple("death"),[]))]
    | lex "decline" = [li(syn("TV"),cc(simple("decline"),[]))]
    | lex "declines" = [li(syn("TV"),cc(simple("decline"),[]))]
    | lex "dependent" = [li(syn("A"),cc(simple("dependent"),[]))]
    | lex "detected" = [li(syn("VBN"),cc(simple("detection"),[]))]
    | lex "detection" = [li(syn("N"),cc(simple("detection"),[]))]
    | lex "determined" = [li(syn("VBD"),cc(simple("determining"),[])),li(syn("VBN"),cc(simple("determining"),[]))]
    | lex "disposal" = [li(syn("N"),cc(simple("disposal"),[]))]
    | lex "dna" = [li(syn("N"),cc(simple("dna"),[]))]
    | lex "effect" = [li(syn("N"),cc(simple("effect"),[]))]
    | lex "elevation" = [li(syn("N"),cc(simple("elevation"),[]))]
    | lex "enters" = [li(syn("TV"),cc(simple("entry"),[]))]
    | lex "entry" = [li(syn("N"),cc(simple("entery"),[]))]
    | lex "excited" = [li(syn("VBD"),cc(simple("excitement"),[])),li(syn("VBN"),cc(simple("excitement"),[]))]
    | lex "excrete" = [li(syn("TV"),cc(simple("excretion"),[]))]
    | lex "excretes" = [li(syn("TV"),cc(simple("excretion"),[]))]
    | lex "excreting" = [li(syn("VBG"),cc(simple("excretion"),[]))]
    | lex "excursion" = [li(syn("N"),cc(simple("excursion"),[]))]
    | lex "express" = [li(syn("TV"),cc(simple("expression"),[]))]
    | lex "facilitated" = [li(syn("VBD"),cc(simple("facilitation"),[])),li(syn("VBN"),cc(simple("facilitation"),[]))]
    | lex "fasting" = [li(syn("VBG"),cc(simple("fasting"),[]))]
    | lex "fat" = [li(syn("N"),cc(simple("fat"),[]))]
    | lex "fed" = [li(syn("VBD"),cc(simple("feeding"),[])),li(syn("VBN"),cc(simple("feeding"),[]))]
    | lex "formed" = [li(syn("VBD"),cc(simple("forming"),[])),li(syn("VBN"),cc(simple("forming"),[]))]
```

```
    | lex "function" = [li(syn("N"),cc(simple("function"),[])),li(
      syn("TV"),cc(simple("functioning"),[]))]
    | lex "generates" = [li(syn("TV"),cc(simple("generation"),[]))
      ]
    | lex "glucagon" = [li(syn("N"),cc(simple("glucagon"),[]))]
    | lex "glucokinase" = [li(syn("N"),cc(simple("glucokinase")
      ,[]))]
109 | lex "gluconeogenesis" = [li(syn("N"),cc(simple("
      gluconeogenesis"),[]))]
    | lex "glucose" = [li(syn("N"),cc(simple("glucose"),[]))]
    | lex "glucose-6-phosphatase" = [li(syn("N"),cc(simple("
      glucose_6_phosphatase"),[]))]
    | lex "glucosensing" = [li(syn("N"),cc(simple("glucosensing")
      ,[]))]
    | lex "glut2" = [li(syn("N"),cc(simple("glut2"),[]))]
114 | lex "glut4" = [li(syn("N"),cc(simple("glut4"),[]))]
    | lex "glycogen" = [li(syn("N"),cc(simple("glycogen"),[]))]
    | lex "glycogenolysis" = [li(syn("N"),cc(simple("
      glycogenolysis"),[]))]
    | lex "glycoprotein" = [li(syn("N"),cc(simple("glycoprotein")
      ,[]))]
    | lex "growth" = [li(syn("N"),cc(simple("growth"),[]))]
119 | lex "high" = [li(syn("A"),cc(simple("high"),[]))]
    | lex "hsp27" = [li(syn("N"),cc(simple("hsp27"),[]))]
    | lex "impaired" = [li(syn("VBD"),cc(simple("impairment"),[]))
      ,li(syn("VBN"),cc(simple("impairment"),[]))]
    | lex "include" = [li(syn("TV"),cc(simple("inclusion"),[]))]
    | lex "incorporation" = [li(syn("N"),cc(simple("incorporation"
      ),[]))]
124 | lex "increase" = [li(syn("N"),cc(simple("increase"),[]))]
    | lex "increased" = [li(syn("VBD"),cc(simple("increase"),[])),
      li(syn("VBN"),cc(simple("increased"),[]))]
    | lex "incubation" = [li(syn("N"),cc(simple("incubation"),[]))
      ]
    | lex "indicates" = [li(syn("TV"),cc(simple("indication"),[]))
      ]
    | lex "induce" = [li(syn("TV"),cc(simple("induction"),[]))]
129 | lex "induced" = [li(syn("VBD"),cc(simple("induction"),[])),
      li(syn("VBN"),cc(simple("induction"),[]))]
```

```
    | lex "induces" = [li(syn("TV"),cc(simple("induction"),[]))]
    | lex "inducing" = [li(syn("VBG"),cc(simple("induction"),[]))]
    | lex "induction" = [li(syn("N"),cc(simple("induction"),[]))]
    | lex "inhibit" = [li(syn("TV"),cc(simple("inhibition"),[]))]
    | lex "inhibited" = [li(syn("VBD"),cc(simple("inhibition"),[])
      ),li(syn("VBN"),cc(simple("inhibition"),[]))]
    | lex "inhibiting" = [li(syn("VBG"),cc(simple("inhibition")
      ,[]))]
    | lex "inhibition" = [li(syn("N"),cc(simple("inhibition"),[]))
      ]
    | lex "inhibitor" = [li(syn("N"),cc(simple("inhibitor"),[]))]
    | lex "inhibitors" = [li(syn("N"),cc(simple("inhibitor"),[]))]
    | lex "inhibits" = [li(syn("TV"),cc(simple("inhibition"),[]))]
    | lex "insulin" = [li(syn("N"),cc(simple("insulin"),[]))]
    | lex "insulinotropic" = [li(syn("A"),cc(simple("
      insulinotropic"),[]))]
    | lex "involved" = [li(syn("VBD"),cc(simple("involvment"),[]))
      ,li(syn("VBN"),cc(simple("involvment"),[]))]
    | lex "is" = [li(syn("IS"),cc(simple("is"),[]))]
    | lex "john" = [li(syn("N"),cc(simple("john"),[]))]
    | lex "kit" = [li(syn("N"),cc(simple("kit"),[]))]
    | lex "level" = [li(syn("N"),cc(simple("level"),[]))]
    | lex "levels" = [li(syn("N"),cc(simple("level"),[]))]
    | lex "liver" = [li(syn("N"),cc(simple("liver"),[]))]
    | lex "low" = [li(syn("A"),cc(simple("low"),[]))]
    | lex "lowers" = [li(syn("TV"),cc(simple("decrease"),[]))]
    | lex "manifestations" = [li(syn("N"),cc(simple("manifestation
      "),[]))]
    | lex "may" = [li(syn("CAN"),cc(simple("can"),[]))]
    | lex "mediated" = [li(syn("VBD"),cc(simple("mediation"),[])),
      li(syn("VBN"),cc(simple("mediation"),[]))]
    | lex "mediator" = [li(syn("N"),cc(simple("mediator"),[]))]
    | lex "membrane" = [li(syn("N"),cc(simple("membrane"),[]))]
    | lex "membranes" = [li(syn("N"),cc(simple("membrane"),[]))]
    | lex "mixture" = [li(syn("N"),cc(simple("mixture"),[]))]
    | lex "molecular" = [li(syn("A"),cc(simple("molecular"),[]))]
    | lex "molecule" = [li(syn("N"),cc(simple("molecule"),[]))]
    | lex "muscle" = [li(syn("N"),cc(simple("muscle"),[]))]
    | lex "neurons" = [li(syn("N"),cc(simple("neuron"),[]))]
```

```
    | lex "organism" = [li(syn("N"),cc(simple("organism"),[]))]
    | lex "oxidation" = [li(syn("N"),cc(simple("oxidation"),[]))]
    | lex "oxygen" = [li(syn("N"),cc(simple("oxygen"),[]))]
    | lex "pathway" = [li(syn("N"),cc(simple("pathway"),[]))]
    | lex "peripheral" = [li(syn("A"),cc(simple("peripheral"),[]))
      ]
    | lex "pgg" = [li(syn("N"),cc(simple("pgg"),[]))]
    | lex "phosphorylation" = [li(syn("N"),cc(simple("
      phosphorylation"),[]))]
    | lex "pkc" = [li(syn("N"),cc(simple("pkc"),[]))]
    | lex "plasma" = [li(syn("N"),cc(simple("plasma"),[]))]
    | lex "polypeptide" = [li(syn("N"),cc(simple("polypeptide")
      ,[]))]
    | lex "potentiates" = [li(syn("TV"),cc(simple("potentiation")
      ,[]))]
    | lex "production" = [li(syn("N"),cc(simple("production"),[]))
      ]
    | lex "prolonged" = [li(syn("VBN"),cc(simple("prolongation")
      ,[]))]
    | lex "promote" = [li(syn("TV"),cc(simple("promotion"),[]))]
    | lex "promotes" = [li(syn("TV"),cc(simple("promotion"),[]))]
    | lex "protein" = [li(syn("N"),cc(simple("protein"),[]))]
    | lex "receptor" = [li(syn("N"),cc(simple("receptor"),[]))]
    | lex "regulatable" = [li(syn("A"),cc(simple("regulatable")
      ,[]))]
    | lex "regulated" = [li(syn("VBD"),cc(simple("regulation"),[])
      ),li(syn("VBN"),cc(simple("regulation"),[]))]
    | lex "regulation" = [li(syn("N"),cc(simple("regulation"),[]))
      ]
    | lex "release" = [li(syn("N"),cc(simple("release"),[]))]
    | lex "replication" = [li(syn("N"),cc(simple("replication")
      ,[]))]
    | lex "require" = [li(syn("TV"),cc(simple("requirement"),[]))]
    | lex "requires" = [li(syn("TV"),cc(simple("requirement"),[]))
      ]
    | lex "resistance" = [li(syn("N"),cc(simple("resistance"),[]))
      ]
    | lex "responses" = [li(syn("N"),cc(simple("response"),[]))]
```

```
    | lex "responsive" = [li(syn("A"),cc(simple("responsive"),[]))
      ]
189 | lex "resulting" = [li(syn("VBG"),cc(simple("result"),[]))]
    | lex "secreted" = [li(syn("VBD"),cc(simple("secretion"),[])),
      li(syn("VBN"),cc(simple("secretion"),[]))]
    | lex "secretes" = [li(syn("TV"),cc(simple("secretion"),[]))]
    | lex "secreting" = [li(syn("VBG"),cc(simple("secretion"),[]))
      ]
    | lex "secretion" = [li(syn("N"),cc(simple("secretion"),[]))]
194 | lex "secretory" = [li(syn("A"),cc(simple("secretory"),[]))]
    | lex "sensed" = [li(syn("VBD"),cc(simple("sensing"),[])),li(
      syn("VBN"),cc(simple("sensing"),[]))]
    | lex "sensitive" = [li(syn("A"),cc(simple("sensitive"),[]))]
    | lex "serum" = [li(syn("N"),cc(simple("serum"),[]))]
    | lex "signaling" = [li(syn("VBG"),cc(simple("signaling"),[]))
      ]
199 | lex "smaller" = [li(syn("A"),cc(simple("smaller"),[]))]
    | lex "somatostatin" = [li(syn("N"),cc(simple("somatostatin")
      ,[]))]
    | lex "specific" = [li(syn("A"),cc(simple("specific"),[]))]
    | lex "stimulate" = [li(syn("TV"),cc(simple("stimulation"),[])
      )]
    | lex "stimulated" = [li(syn("VBD"),cc(simple("stimulation")
      ,[])),li(syn("VBN"),cc(simple("stimulation"),[]))]
204 | lex "stimulates" = [li(syn("TV"),cc(simple("stimulation")
      ,[]))]
    | lex "stimulating" = [li(syn("VBG"),cc(simple("stimulation")
      ,[]))]
    | lex "synthase" = [li(syn("N"),cc(simple("synthase"),[]))]
    | lex "synthesis" = [li(syn("N"),cc(simple("synthesis"),[]))]
    | lex "tissues" = [li(syn("N"),cc(simple("tissue"),[]))]
209 | lex "tolerance" = [li(syn("N"),cc(simple("tolerance"),[]))]
    | lex "translocation" = [li(syn("N"),cc(simple("translocation"
      ),[]))]
    | lex "transport" = [li(syn("N"),cc(simple("transport"),[])),
      li(syn("TV"),cc(simple("transport"),[]))]
    | lex "transportation" = [li(syn("N"),cc(simple("transport")
      ,[]))]
```

```
    | lex "transported" = [li(syn("VBD"),cc(simple("transport")
      ,[])),li(syn("VBN"),cc(simple("transport"),[]))]
    | lex "transporter" = [li(syn("N"),cc(simple("transporter")
      ,[]))]
    | lex "transporters" = [li(syn("N"),cc(simple("transporter")
      ,[]))]
    | lex "transporting" = [li(syn("VBG"),cc(simple("transport")
      ,[]))]
    | lex "transports" = [li(syn("TV"),cc(simple("transport"),[]))
      ]
    | lex "tyrosine" = [li(syn("N"),cc(simple("tyrosine"),[]))]
    | lex "uptake" = [li(syn("N"),cc(simple("uptake"),[]))]
    | lex "utilization" = [li(syn("N"),cc(simple("utilization")
      ,[]))]
    | lex "vasopressin" = [li(syn("N"),cc(simple("vasopressin")
      ,[]))]
    | lex "wall" = [li(syn("N"),cc(simple("wall"),[]))]
    | lex "was" = [li(syn("IS"),cc(simple("is"),[]))]
    | lex "weigth" = [li(syn("N"),cc(simple("weigth"),[]))]
    | lex "were" = [li(syn("IS"),cc(simple("is"),[]))]
    | lex "zcytor17lig" = [li(syn("N"),cc(simple("zcytor17lig")
      ,[]))]
    | lex w = raise lex_not_found w

  fun isa (simple "abolition") = [simple "abolition",simple "event
    ",simple "influence",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "univ"]
    | isa (simple "acinar_cell") = [simple "acinar_cell",simple "
      anatomical_structure",simple "cell",simple "entity",simple
      "exocrine_cell",simple "exocrine_pancreatic_cell",simple "
      fully_formed_anatomical_structure",simple "pancreatic_cell"
      ,simple "physical_object",simple "univ"]
    | isa (simple "activation") = [simple "activation",simple "
      event",simple "influence",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "univ"]
    | isa (simple "activity") = [simple "activity",simple "event",
      simple "natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "univ"]
```

```
  | isa (simple "addition") = [simple "addition",simple "event",
    simple "phenomenon_or_process",simple "univ"]
  | isa (simple "adipocyte") = [simple "adipocyte",simple "
    anatomical_structure",simple "cell",simple "entity",simple
    "fully_formed_anatomical_structure",simple "physical_object
    ",simple "univ"]
  | isa (simple "adult_stem_cell") = [simple "adult_stem_cell",
    simple "anatomical_structure",simple "cell",simple "entity"
    ,simple "fully_formed_anatomical_structure",simple "
    physical_object",simple "stem_cell",simple "univ"]
  | isa (simple "agent") = [simple "agent",simple "
    conceptual_entity",simple "entity",simple "substance",
    simple "univ"]
  | isa (simple "allimin") = [simple "allimin",simple "
    conceptual_entity",simple "entity",simple "protein",simple
    "substance",simple "univ"]
  | isa (simple "alpha_cell") = [simple "alpha_cell",simple "
    anatomical_structure",simple "cell",simple "endocrine_cell"
    ,simple "endocrine_pancreatic_cell",simple "entity",simple
    "fully_formed_anatomical_structure",simple "pancreatic_cell
    ",simple "physical_object",simple "univ"]
  | isa (simple "aminoacid") = [simple "aminoacid",simple "
    conceptual_entity",simple "entity",simple "substance",
    simple "univ"]
  | isa (simple "anatomical_structure") = [simple "
    anatomical_structure",simple "entity",simple "
    physical_object",simple "univ"]
  | isa (simple "antifungal") = [simple "antifungal",simple "
    property",simple "substance_property",simple "univ"]
  | isa (simple "betacell") = [simple "anatomical_structure",
    simple "betacell",simple "cell",simple "endocrine_cell",
    simple "endocrine_pancreatic_cell",simple "entity",simple "
    fully_formed_anatomical_structure",simple "pancreatic_cell"
    ,simple "physical_object",simple "univ"]
  | isa (simple "biologic_function") = [simple "
    biologic_function",simple "event",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
```

```
244     | isa (simple "blood") = [simple "anatomical_structure",simple
          "blood",simple "body_part_or_organ_or_organ_component",
        simple "entity",simple "fully_formed_anatomical_structure",
        simple "organ",simple "physical_object",simple "univ"]
    | isa (simple "body_part_or_organ_or_organ_component") = [
        simple "anatomical_structure",simple "
        body_part_or_organ_or_organ_component",simple "entity",
        simple "fully_formed_anatomical_structure",simple "
        physical_object",simple "univ"]
    | isa (simple "body_system") = [simple "body_system",simple "
        conceptual_entity",simple "entity",simple "
        functional_concept",simple "idea_or_concept",simple "
        physical_object",simple "univ"]
    | isa (simple "c_peptide") = [simple "c_peptide",simple "
        conceptual_entity",simple "entity",simple "substance",
        simple "univ"]
    | isa (simple "carrier") = [simple "carrier",simple "
        conceptual_entity",simple "entity",simple "substance",
        simple "transporter",simple "univ"]
249 | isa (simple "caspase") = [simple "caspase",simple "
        conceptual_entity",simple "entity",simple "protein",simple
        "substance",simple "univ"]
    | isa (simple "causing") = [simple "causing",simple "event",
        simple "influence",simple "natural_phenomenon_or_process",
        simple "phenomenon_or_process",simple "univ"]
    | isa (simple "cell") = [simple "anatomical_structure",simple
        "cell",simple "entity",simple "
        fully_formed_anatomical_structure",simple "physical_object"
        ,simple "univ"]
    | isa (simple "central") = [simple "central",simple "property"
        ,simple "structure_property",simple "univ"]
    | isa (simple "centroacinar_cell") = [simple "
        anatomical_structure",simple "cell",simple "
        centroacinar_cell",simple "entity",simple "exocrine_cell",
        simple "exocrine_pancreatic_cell",simple "
        fully_formed_anatomical_structure",simple "pancreatic_cell"
        ,simple "physical_object",simple "univ"]
254 | isa (simple "challenge") = [simple "challenge",simple "event
        ",simple "phenomenon_or_process",simple "univ"]
```

```
  | isa (simple "change") = [simple "change",simple "event",
    simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "cloning") = [simple "cloning",simple "event",
    simple "phenomenon_or_process",simple "univ"]
  | isa (simple "combination") = [simple "combination"]
  | isa (simple "compound") = [simple "compound",simple "
    conceptual_entity",simple "entity",simple "substance",
    simple "univ"]
  | isa (simple "concentration") = [simple "concentration",
    simple "level"]
  | isa (simple "conceptual_entity") = [simple "
    conceptual_entity",simple "entity",simple "univ"]
  | isa (simple "control") = [simple "control",simple "event",
    simple "influence",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "univ"]
  | isa (simple "death") = [simple "death",simple "event",simple
     "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "decline") = [simple "change",simple "decline",
    simple "event",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "univ"]
  | isa (simple "decrease") = [simple "decrease",simple "event",
    simple "influence",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "univ"]
  | isa (simple "dependent") = [simple "dependent",simple "
    process_property",simple "property",simple "univ"]
  | isa (simple "detection") = [simple "detection",simple "event
    ",simple "medical_procedure",simple "phenomenon_or_process"
    ,simple "univ"]
  | isa (simple "disposal") = [simple "disposal",simple "event",
    simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "production",simple "
    substance_process",simple "univ"]
  | isa (simple "dna") = [simple "conceptual_entity",simple "dna
    ",simple "entity",simple "substance",simple "univ"]
  | isa (simple "effect") = [simple "effect",simple "event",
    simple "phenomenon_or_process",simple "univ"]
```

```
  | isa (simple "embryonic_stem_cell") = [simple "
    anatomical_structure",simple "cell",simple "
    embryonic_stem_cell",simple "entity",simple "
    fully_formed_anatomical_structure",simple "physical_object"
    ,simple "stem_cell",simple "univ"]
  | isa (simple "endocrine_cell") = [simple "
    anatomical_structure",simple "cell",simple "endocrine_cell"
    ,simple "entity",simple "fully_formed_anatomical_structure"
    ,simple "physical_object",simple "univ"]
  | isa (simple "endocrine_pancreatic_cell") = [simple "
    anatomical_structure",simple "cell",simple "endocrine_cell"
    ,simple "endocrine_pancreatic_cell",simple "entity",simple
    "fully_formed_anatomical_structure",simple "pancreatic_cell
    ",simple "physical_object",simple "univ"]
  | isa (simple "entity") = [simple "entity",simple "univ"]
  | isa (simple "entry") = [simple "entry",simple "event",simple
    "phenomenon_or_process",simple "univ"]
  | isa (simple "enzyme") = [simple "conceptual_entity",simple "
    entity",simple "enzyme",simple "substance",simple "univ"]
  | isa (simple "event") = [simple "event",simple "univ"]
  | isa (simple "excitement") = [simple "event",simple "
    excitement",simple "influence",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "excretion") = [simple "event",simple "excretion
    ",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "production",simple "
    substance_process",simple "univ"]
  | isa (simple "excursion") = [simple "event",simple "excursion
    ",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "substance_process",simple "
    univ"]
  | isa (simple "exocrine_cell") = [simple "anatomical_structure
    ",simple "cell",simple "entity",simple "exocrine_cell",
    simple "fully_formed_anatomical_structure",simple "
    physical_object",simple "univ"]
  | isa (simple "exocrine_pancreatic_cell") = [simple "
    anatomical_structure",simple "cell",simple "entity",simple
    "exocrine_cell",simple "exocrine_pancreatic_cell",simple "
```

```
    fully_formed_anatomical_structure",simple "pancreatic_cell"
    ,simple "physical_object",simple "univ"]
| isa (simple "facilitation") = [simple "event",simple "
    facilitation",simple "influence",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
| isa (simple "fat") = [simple "anatomical_structure",simple "
    body_part_or_organ_or_organ_component",simple "entity",
    simple "fat",simple "fully_formed_anatomical_structure",
    simple "physical_object",simple "tissue",simple "univ"]
| isa (simple "fully_formed_anatomical_structure") = [simple "
    anatomical_structure",simple "entity",simple "
    fully_formed_anatomical_structure",simple "physical_object"
    ,simple "univ"]
| isa (simple "functional_concept") = [simple "
    conceptual_entity",simple "entity",simple "
    functional_concept",simple "idea_or_concept",simple "univ"]
| isa (simple "gamma_cell") = [simple "anatomical_structure",
    simple "cell",simple "endocrine_cell",simple "
    endocrine_pancreatic_cell",simple "entity",simple "
    fully_formed_anatomical_structure",simple "gamma_cell",
    simple "pancreatic_cell",simple "physical_object",simple "
    univ"]
| isa (simple "general_property") = [simple "general_property"
    ]
| isa (simple "glucagon") = [simple "conceptual_entity",simple
    "entity",simple "glucagon",simple "substance",simple "univ
    "]
| isa (simple "glucokinase") = [simple "conceptual_entity",
    simple "entity",simple "enzyme",simple "glucokinase",simple
    "substance",simple "univ"]
| isa (simple "gluconeogenesis") = [simple "event",simple "
    gluconeogenesis",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "univ"]
| isa (simple "glucose") = [simple "conceptual_entity",simple
    "entity",simple "glucose",simple "substance",simple "univ"]
| isa (simple "glucose_6_phosphatase") = [simple "
    conceptual_entity",simple "entity",simple "
    glucose_6_phosphatase",simple "substance",simple "univ"]
```

```
    | isa (simple "glucose_excited") = [simple "glucose_excited",
      simple "property",simple "structure_property",simple "univ"
      ]
294 | isa (simple "glucosensing") = [simple "event",simple "
      glucosensing",simple "natural_phenomenon_or_process",simple
       "phenomenon_or_process",simple "univ"]
    | isa (simple "glut2") = [simple "conceptual_entity",simple "
      entity",simple "glut2",simple "substance",simple "
      transporter",simple "univ"]
    | isa (simple "glut4") = [simple "conceptual_entity",simple "
      entity",simple "glut4",simple "substance",simple "
      transporter",simple "univ"]
    | isa (simple "glycogen") = [simple "conceptual_entity",simple
       "entity",simple "glycogen",simple "molecule",simple "
      substance",simple "univ"]
    | isa (simple "glycogenolysis") = [simple "event",simple "
      glycogenolysis",simple "natural_phenomenon_or_process",
      simple "phenomenon_or_process",simple "univ"]
299 | isa (simple "glycoprotein") = [simple "conceptual_entity",
      simple "entity",simple "glycoprotein",simple "protein",
      simple "substance",simple "univ"]
    | isa (simple "growth") = [simple "biologic_function",simple "
      event",simple "growth",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "univ"]
    | isa (simple "high") = [simple "high",simple "
      position_on_scale"]
    | isa (simple "hsp27") = [simple "conceptual_entity",simple "
      entity",simple "hsp27",simple "substance",simple "univ"]
    | isa (simple "idea_or_concept") = [simple "conceptual_entity"
      ,simple "entity",simple "idea_or_concept",simple "univ"]
304 | isa (simple "incorporation") = [simple "event",simple "
      incorporation",simple "natural_phenomenon_or_process",
      simple "phenomenon_or_process",simple "substance_process",
      simple "univ"]
    | isa (simple "increase") = [simple "event",simple "increase",
      simple "influence",simple "natural_phenomenon_or_process",
      simple "phenomenon_or_process",simple "univ"]
```

```
  | isa (simple "increased") = [simple "increased",simple "
    position_on_scale"]
  | isa (simple "incubation") = [simple "event",simple "
    incubation",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "induction") = [simple "event",simple "induction
    ",simple "influence",simple "natural_phenomenon_or_process"
    ,simple "phenomenon_or_process",simple "univ"]
  | isa (simple "influence") = [simple "event",simple "influence
    ",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "inhibition") = [simple "event",simple "
    influence",simple "inhibition",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "inhibitor") = [simple "conceptual_entity",
    simple "entity",simple "inhibitor",simple "substance",
    simple "univ"]
  | isa (simple "insulin") = [simple "conceptual_entity",simple
    "entity",simple "insulin",simple "protein",simple "
    substance",simple "univ"]
  | isa (simple "insulin_regulatable") = [simple "
    insulin_regulatable",simple "property",simple "
    substance_property",simple "univ"]
  | isa (simple "insulinotropic") = [simple "insulinotropic",
    simple "property",simple "substance_property",simple "univ"
    ]
  | isa (simple "is") = [simple "is"]
  | isa (simple "level") = [simple "level"]
  | isa (simple "liver") = [simple "anatomical_structure",simple
     "body_part_or_organ_or_organ_component",simple "entity",
    simple "fully_formed_anatomical_structure",simple "liver",
    simple "organ",simple "physical_object",simple "univ"]
  | isa (simple "low") = [simple "low",simple "position_on_scale
    "]
  | isa (simple "medical_procedure") = [simple "event",simple "
    medical_procedure",simple "phenomenon_or_process",simple "
    univ"]
```

```
  | isa (simple "membrane") = [simple "anatomical_structure",
    simple "entity",simple "membrane",simple "physical_object",
    simple "univ"]
  | isa (simple "mixture") = [simple "combination",simple "
    mixture"]
  | isa (simple "molecular") = [simple "molecular",simple "
    process_property",simple "property",simple "univ"]
  | isa (simple "molecule") = [simple "conceptual_entity",simple
     "entity",simple "molecule",simple "substance",simple "univ
    "]
  | isa (simple "muscle") = [simple "anatomical_structure",
    simple "body_part_or_organ_or_organ_component",simple "
    entity",simple "fully_formed_anatomical_structure",simple "
    muscle",simple "organ",simple "physical_object",simple "
    univ"]
  | isa (simple "natural_phenomenon_or_process") = [simple "
    event",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "univ"]
  | isa (simple "nervous_system") = [simple "body_system",simple
     "conceptual_entity",simple "entity",simple "
    functional_concept",simple "idea_or_concept",simple "
    nervous_system",simple "physical_object",simple "univ"]
  | isa (simple "neuron") = [simple "anatomical_structure",
    simple "cell",simple "entity",simple "
    fully_formed_anatomical_structure",simple "neuron",simple "
    physical_object",simple "univ"]
  | isa (simple "neuronal_schwann_cell") = [simple "
    anatomical_structure",simple "cell",simple "entity",simple
    "fully_formed_anatomical_structure",simple "
    neuronal_schwann_cell",simple "physical_object",simple "
    univ"]
  | isa (simple "organ") = [simple "anatomical_structure",simple
     "body_part_or_organ_or_organ_component",simple "entity",
    simple "fully_formed_anatomical_structure",simple "organ",
    simple "physical_object",simple "univ"]
  | isa (simple "organism") = [simple "anatomical_structure",
    simple "entity",simple "fully_formed_anatomical_structure",
    simple "organism",simple "physical_object",simple "univ"]
```

```
  | isa (simple "oxidation") = [simple "event",simple "
      natural_phenomenon_or_process",simple "oxidation",simple "
      phenomenon_or_process",simple "substance_process",simple "
      univ"]
  | isa (simple "oxygen") = [simple "conceptual_entity",simple "
      entity",simple "oxygen",simple "substance",simple "univ"]
  | isa (simple "pancreatic_cell") = [simple "
      anatomical_structure",simple "cell",simple "entity",simple
      "fully_formed_anatomical_structure",simple "pancreatic_cell
      ",simple "physical_object",simple "univ"]
  | isa (simple "pathway") = [simple "conceptual_entity",simple
      "entity",simple "pathway",simple "univ"]
  | isa (simple "pgg") = [simple "conceptual_entity",simple "
      entity",simple "pgg",simple "substance",simple "univ"]
  | isa (simple "phenomenon_or_process") = [simple "event",
      simple "phenomenon_or_process",simple "univ"]
  | isa (simple "phosphorylation") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "phosphorylation",simple "
      univ"]
  | isa (simple "physical_object") = [simple "entity",simple "
      physical_object",simple "univ"]
  | isa (simple "pkc") = [simple "conceptual_entity",simple "
      entity",simple "pkc",simple "substance",simple "univ"]
  | isa (simple "polynucleotide") = [simple "conceptual_entity",
      simple "entity",simple "polynucleotide",simple "substance",
      simple "univ"]
  | isa (simple "polypeptide") = [simple "conceptual_entity",
      simple "entity",simple "polypeptide",simple "substance",
      simple "univ"]
  | isa (simple "position_on_scale") = [simple "
      position_on_scale"]
  | isa (simple "potentiation") = [simple "event",simple "
      influence",simple "natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "potentiation",simple "univ"]
  | isa (simple "pp_cell") = [simple "anatomical_structure",
      simple "cell",simple "endocrine_cell",simple "
      endocrine_pancreatic_cell",simple "entity",simple "
      fully_formed_anatomical_structure",simple "pancreatic_cell"
```

```
    ,simple "physical_object",simple "pp_cell",simple "univ"]
  | isa (simple "process_property") = [simple "process_property"
    ,simple "property",simple "univ"]
  | isa (simple "production") = [simple "event",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "production",simple "
    substance_process",simple "univ"]
  | isa (simple "promotion") = [simple "event",simple "influence
    ",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "promotion",simple "univ"]
  | isa (simple "property") = [simple "property",simple "univ"]
  | isa (simple "protein") = [simple "conceptual_entity",simple
    "entity",simple "protein",simple "substance",simple "univ"]
  | isa (simple "q_3t3l1") = [simple "anatomical_structure",
    simple "cell",simple "entity",simple "
    fully_formed_anatomical_structure",simple "physical_object"
    ,simple "q_3t3l1",simple "univ"]
  | isa (simple "receptor") = [simple "anatomical_structure",
    simple "entity",simple "fully_formed_anatomical_structure",
    simple "physical_object",simple "receptor",simple "univ"]
  | isa (simple "regulation") = [simple "event",simple "
    influence",simple "natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "regulation",simple "univ"]
  | isa (simple "release") = [simple "event",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "production",simple "release"
    ,simple "substance_process",simple "univ"]
  | isa (simple "replication") = [simple "event",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "production",simple "
    replication",simple "substance_process",simple "univ"]
  | isa (simple "requirement") = [simple "event",simple "
    natural_phenomenon_or_process",simple "
    phenomenon_or_process",simple "requirement",simple "
    substance_process",simple "univ"]
  | isa (simple "resistance") = [simple "biologic_function",
    simple "event",simple "natural_phenomenon_or_process",
    simple "phenomenon_or_process",simple "resistance",simple "
    univ"]
```

```
  | isa (simple "response") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "response",simple "univ"]
  | isa (simple "secretion") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "production",simple "
      secretion",simple "substance_process",simple "univ"]
359 | isa (simple "secretory") = [simple "process_property",simple
       "property",simple "secretory",simple "univ"]
  | isa (simple "sensing") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "sensing",simple "
      substance_process",simple "univ"]
  | isa (simple "signal") = [simple "conceptual_entity",simple "
      entity",simple "signal",simple "univ"]
  | isa (simple "signaling") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "signaling",simple "
      substance_process",simple "univ"]
  | isa (simple "smaller") = [simple "position_on_scale",simple
      "smaller"]
364 | isa (simple "somatostatin") = [simple "conceptual_entity",
      simple "entity",simple "somatostatin",simple "substance",
      simple "univ"]
  | isa (simple "specific") = [simple "general_property",simple
      "property",simple "specific",simple "substance_property",
      simple "univ"]
  | isa (simple "stem_cell") = [simple "anatomical_structure",
      simple "cell",simple "entity",simple "
      fully_formed_anatomical_structure",simple "physical_object"
      ,simple "stem_cell",simple "univ"]
  | isa (simple "stimulation") = [simple "event",simple "
      influence",simple "natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "stimulation",simple "univ"]
  | isa (simple "structure_property") = [simple "property",
      simple "structure_property",simple "univ"]
369 | isa (simple "substance") = [simple "conceptual_entity",
      simple "entity",simple "substance",simple "univ"]
```

```
    | isa (simple "substance_process") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      univ"]
    | isa (simple "substance_property") = [simple "property",
      simple "substance_property",simple "univ"]
    | isa (simple "synthase") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      synthase",simple "univ"]
    | isa (simple "synthesis") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "production",simple "
      substance_process",simple "synthesis",simple "univ"]
374 | isa (simple "tissue") = [simple "anatomical_structure",
      simple "body_part_or_organ_or_organ_component",simple "
      entity",simple "fully_formed_anatomical_structure",simple "
      physical_object",simple "tissue",simple "univ"]
    | isa (simple "tolerance") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      tolerance",simple "univ"]
    | isa (simple "translocation") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      translocation",simple "univ"]
    | isa (simple "transport") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      transport",simple "univ"]
    | isa (simple "transporter") = [simple "conceptual_entity",
      simple "entity",simple "substance",simple "transporter",
      simple "univ"]
379 | isa (simple "tyrosine") = [simple "aminoacid",simple "
      conceptual_entity",simple "entity",simple "substance",
      simple "tyrosine",simple "univ"]
    | isa (simple "univ") = [simple "univ"]
    | isa (simple "uptake") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
```

```
      phenomenon_or_process",simple "substance_process",simple "
      univ",simple "uptake"]
  | isa (simple "utilization") = [simple "event",simple "
      natural_phenomenon_or_process",simple "
      phenomenon_or_process",simple "substance_process",simple "
      univ",simple "utilization"]
  | isa (simple "vasopressin") = [simple "conceptual_entity",
      simple "entity",simple "substance",simple "univ",simple "
      vasopressin"]
  | isa (simple "wall") = [simple "anatomical_structure",simple
      "entity",simple "fully_formed_anatomical_structure",simple
      "physical_object",simple "univ",simple "wall"]
  | isa (simple "weigth") = [simple "property",simple "univ",
      simple "weigth"]
  | isa (simple "zcytor17lig") = [simple "conceptual_entity",
      simple "entity",simple "polynucleotide",simple "substance",
      simple "univ",simple "zcytor17lig"]
  | isa (simple x) = [simple x]

fun isa_chk (x,y) = List.exists (fn z=>z=y) (isa x)

val a_data = [(simple "DUMMY",r "dum",simple "DUMMY"),
              (simple "influence",r "agt",simple "substance"),
              (simple "substance_process",r "pnt",simple "
                 substance"),
              (simple "influence",r "agt",simple "level"),
              (simple "influence",r "pnt",simple "
                 phenomenon_or_process"),
              (simple "influence",r "pnt",simple "
                 anatomical_structure"),
              (simple "physical_object",r "loc",simple "
                 anatomical_structure"),
              (simple "physical_object",r "pof",simple "
                 physical_object"),
              (simple "production",r "agt",simple "
                 anatomical_structure"),
              (simple "phenomenon_or_process",r "loc",simple "
                 anatomical_structure"),
```

```
                (simple "substance",r "src",simple "
                    anatomical_structure"),
                (simple "production",r "src",simple "
                    anatomical_structure"),
                (simple "transport",r "via",simple "
                    anatomical_structure"),
404             (simple "entry",r "agt",simple "substance"),
                (simple "entry",r "pnt",simple "
                    anatomical_structure"),
                (simple "entry",r "via",simple "transporter"),
                (simple "level",r "chr",simple "position_on_scale"
                    ),
                (simple "substance",r "chr",simple "
                    substance_property"),
409             (simple "phenomenon_or_process",r "chr",simple "
                    process_property"),
                (simple "anatomical_structure",r "chr",simple "
                    structure_property"),
                (simple "level",r "wrt",simple "substance"),
                (simple "level",r "loc",simple "blood"),
                (simple "transporter",r "wrt",simple "substance"),
414             (simple "general_property",r "wrt",simple "univ"),
                (simple "is",r "qqq",simple "univ")]

    fun member (x,ys) = List.exists (fn y=>y=x) ys

419 fun aff (sc1,sc2) =
        let
            val sc1ancestors = isa sc1
            val temp0 = List.filter (fn(x,y,z)=>member(x,
                sc1ancestors)) a_data
            val temp1 = List.filter (fn(x,y,z)=>isa_chk(sc2,z))
                temp0
424         val roles = List.map (fn(x,y,z)=>y) temp1
        in
            Binaryset.listItems(Binaryset.addList(Binaryset.empty
                role_compare,roles))
        end
```

```
429 end
```

Appendix F

SML implementation of the ontograbber

Below we present the full listing of an implementation of the ontograbber in the Standard Meta Language. This is the ontograbber presented in Chapter 8 on page 141.

```
load "GenerativeOntology";
load "Listsort";
load "Substring";
open GenerativeOntology;

type cover_begin = int
type cover_end = int

datatype cover = e of cover_begin * cover_end * syntactic_category
    * concept * string list

val i2s = Int.toString
fun glue [] = ""
  | glue [w] = w
  | glue (w::ws) = w^" "^glue ws

fun e2s (e(c_b,c_e,synt,c,str)) =
    let val v = c2s c
```

```
        val sep = if length(explode v)>20 then "\\\\\n&&\\phantom
            {()" else ""
    in
        "&&_("^i2s c_b^","^i2s c_e^",\\textrm{"
        ^sc2s synt^"},"^v^","^sep
        ^"\\textrm{``"^glue str^"''})"
    end

fun eqnarray [] = ""
  | eqnarray [x] = e2s x^""
  | eqnarray (x::xs) = e2s x^",\\\\\n"^eqnarray xs

fun appendToFile string filename =
    let
        val out_stream = TextIO.openAppend filename
    in
        TextIO.output(out_stream,string);
        TextIO.flushOut out_stream;
        TextIO.closeOut out_stream
    end

fun writeToFile string filename =
    let
        val out_stream = TextIO.openOut filename
    in
        TextIO.output(out_stream,string);
        TextIO.flushOut out_stream;
        TextIO.closeOut out_stream
    end

val () = writeToFile "" "progress.tex"

fun a (cc(sc1,_)) (cc(sc2,_)) = aff (sc1,sc2)

(* Set of entries implemented as list *)
type ''a set = ''a list
val empty : cover set = []
fun singleton (item:cover) = [item]
fun add (set:cover set,item:cover)
```

```
  = if List.exists (fn (y:cover)=>y=item) set then set else item::
    set
fun addList (set:cover set,[]) = set
  | addList (set,(x:cover)::xs) = add(addList (set,xs),x)
fun delete (set,item:cover) = List.filter (fn y=>y<>item) set
fun listItems set = set
val isEmpty = List.null
val find = List.find

fun initiate sen =
    let
        fun i [] n (set:cover set) = set
          | i (w::ws) n set =
            let
                fun h ((li(synt,c)):lexicon_item) = ((e(n,n,synt,c
                    ,[w])):cover)
            in
                i ws (n+1) (addList (set,map h (lex w)))
                handle lex_not_found _ => i ws n set
            end
    in
        i sen 0 empty
    end

fun best (cs:cover set) = let val SOME(x) = find (fn _=>true) cs
   in x end

fun addRole (cc(sc1,ps1),role,c2) = cc(sc1,(role,c2)::ps1)

val t_all =
    [fn(str1,syn "VP", c1,r "agt",str2,syn "NP",c2)
      => (str2@str1,syn "S",addRole(c1,r "agt",c2)),
     fn(str1,syn "TV", c1,r "pnt",str2,syn "NP",c2)
      => (str1@str2,syn "VP",addRole(c1,r "pnt",c2)),
     fn(str1,syn "N",  c1,role,str2,syn "N", c2)
      => (str2@str1,syn "N",addRole(c1,role,c2)),
     fn(str1,syn "NP", c1,r "pnt",str2,syn "NP",c2)
      => (str1@["of"]@str2,syn "NP",addRole(c1,r "pnt",c2)),
     fn(str1,syn "NP", c1,r "agt",str2,syn "NP",c2)
```

```
      => (str1@["by"]@str2,syn "NP",addRole(c1,r "agt",c2)),
     fn(str1,syn "NP", c1,r "via",str2,syn "NP",c2)
      => (str1@["across"]@str2,syn "NP",addRole(c1,r "via",c2)),
     fn(str1,syn "NP", c1,r "loc",str2,syn "NP",c2)
      => (str1@["of"]@str2,syn "NP",addRole(c1,r "loc",c2)),
     fn(str1,syn "NP", c1,r "wrt",str2,syn "NP",c2)
      => (str1@["of"]@str2,syn "NP",addRole(c1,r "wrt",c2)),
     fn(str1,syn "NP", c1,r "bmo",str2,syn "NP",c2)
      => (str1@["through"]@str2,syn "NP",addRole(c1,r "bmo",c2)),
     fn(str1,syn "NP", c1,r "cmp",str2,syn "NP",c2)
      => (str1@["of"]@str2,syn "NP",addRole(c1,r "cmp",c2)),
     fn(str1,syn "NP", c1,r "src",str2,syn "NP",c2)
      => (str1@["from"]@str2,syn "NP",addRole(c1,r "src",c2)),
     fn(str1,syn "NP", c1,r "chr",str2,syn "A", c2)
      => (str2@str1,syn "NP",addRole(c1,r "chr",c2)),
     fn(str1,syn "VP", c1,r "via",str2,syn "NP",c2)
      => (str1@str2,syn "VP",addRole(c1,r "via",c2))
    ]

val t_syn_all =
    [fn(str1,syn "NP", c1,str2,syn "NP",c2)
      => (str1@["and"]@str2,syn "NP",c1),
     fn(str1,syn "NP", c1,str2,syn "NP",c2)
      => (str1@["and"]@str2,syn "NP",c2),
     fn(str1,syn "VP", c1,str2,syn "VP",c2)
      => (str1@["and"]@str2,syn "VP",c1),
     fn(str1,syn "VP", c1,str2,syn "VP",c2)
      => (str1@["and"]@str2,syn "VP",c2),
     fn(str1,syn "IS",c1,str2,syn "VBG",c2)
      => (str1@str2,syn "TV",c2)]

fun patterns x [] = []
  | patterns x (t::ts) = (t x)::(patterns x ts) handle Match => (
     patterns x ts)

fun match([],_) = true
  | match(_,[]) = false
  | match(PH::PT,SH::ST) = if PH=SH then match(PT,ST)
                           else match(PH::PT,ST)
```

```
fun combine sentence (e(c_b1,c_e1,s1,c1,str1),
                      e(c_b2,c_e2,s2,c2,str2)) =
    let
        fun n1 role =
            let
                val pats = patterns (str1,s1,c1,role,str2,s2,c2)
                    t_all
                val matched = List.filter (fn (pat,syn',c)=> match
                    (pat,sentence)) pats
            in
                map (fn(pat,syn',c)=>e(c_b1,c_e2,syn',c,pat))
                    matched
            end
        fun n2 role =
            let
                val pats = patterns (str2,s2,c2,role,str1,s1,c1)
                    t_all
                val matched = List.filter (fn (pat,syn',c)=> match
                    (pat,sentence)) pats
            in
                map (fn(pat,syn',c)=>e(c_b1,c_e2,syn',c,pat))
                    matched
            end
    in
        List.concat ((map n1 (a c1 c2)) @ (map n2 (a c2 c1)))
    end

fun syn_combine sentence (e(c_b1,c_e1,s1,c1,str1),
                          e(c_b2,c_e2,s2,c2,str2)) =
    let
        val pats = patterns (str1,s1,c1,str2,s2,c2) t_syn_all
        val matched = List.filter (fn (pat,syn',c)=> match(pat,
            sentence)) pats
    in
        map (fn(pat,syn',c)=>e(c_b1,c_e2,syn',c,pat)) matched
    end

val lift_rule = fn syn "N" => syn "NP" | _ => raise Match
```

```
fun lift (e(c_b,c_e,synt,c,str)) =
    [e(c_b,c_e,lift_rule synt,c,str)] handle Match => []

fun grab sen (C:cover set) (H:cover set) =
    if isEmpty H
    then
        C
    else
        let
            val SOME(h as e(c_b,c_e,_,c,str)) = find (fn _=>true)
                H
            val l_neighbours = List.filter (fn(e(_,y,_,_,_))=>y+1=
                c_b) (listItems C)
            val r_neighbours = List.filter (fn(e(y,_,_,_,_))=>c_e
                +1=y) (listItems C)
            val pairs = (map (fn n=>(n,h)) l_neighbours)@(map (fn
                n=>(h,n)) r_neighbours)
            val lifted = lift h
            val new = lifted
                      @ List.concat (map (fn pair=>combine sen
                          pair) pairs)
                      @ List.concat (map (fn pair=>syn_combine sen
                           pair) pairs)
            val tex = if new=[] then ""
                      else "\n\\begin{eqnarray*}\n"
                           ^"h="^e2s h^"\\\\\n"
                           ^"N=\\{"
                           ^eqnarray new^"\\}\n\\end{eqnarray*}\n"
            val () = appendToFile tex "progress.tex"
        in
            grab sen (addList(C,new)) (delete ((addList(H,new)),h)
                )
        end

fun run sen = let val C_0 = initiate sen in grab sen C_0 C_0 end

fun split s = List.map Substring.string (Substring.fields (fn #" "
    =>true|_=>false) (Substring.all s))
```

```
(* Here are some test sentences *)

val entence = "these low insulin levels are inhibiting the
    transport of glucose and oxygen across cell membranes"
val entence = "these low insulin levels inhibit the transport of
    glucose across cell membranes therefore causing high blood
    glucose levels"
val entence = "glucose enters the beta cells through the glucose
    transporter glut2"
val entence = "insulin also inhibits the release of glucose from
    the liver"
val entence = "the secreted insulin promotes glucose utilization
    and inhibits production of glucose by the liver"
val entence = "three insulin-signaling pathway-specific inhibitors
    also abolish pgg-induced glucose transport in 3t3-l1
    adipocytes"
val sentence = "some anti-fungal agents function as cell wall
    inhibitors by inhibiting glucose synthase"
val entence = "although insulin secretion is predominantly
    controlled by blood levels of glucose , somatostatin inhibits
    glucose-mediated insulin secretory responses"
val entence = "in the liver , insulin inhibits the production of
    glucose by inhibiting gluconeogenesis and glycogenolysis"
val entence = "however , insulin also acts to inhibit the activity
    of glucose-6-phosphatase"
val entence = "the blood glucose levels may be fasting or fed
    glucose levels , and blood glucose levels include serum or
    plasma glucose levels"
val entence = "the insulin secretion determined is preferably
    glucose-stimulated insulin secretion"
val entence = "preferably , the insulin secretion is glucose-
    stimulated insulin secretion"

val t_sen = split sentence

val test = run t_sen

fun bigger_cover (e(c_b,c_e,synt,c,str),(max,strings)) =
```

```
    let val c_dif = c_e-c_b
    in
        if max>c_dif then (max,strings)
        else if max=c_dif then (max,c2s c::strings)
        else (c_dif,[c2s c])
    end

val biggest_cover = foldr bigger_cover (0,[]) test

val C_0 = initiate t_sen

val tex = eqnarray (Listsort.sort (fn(e(a,_,_,_,_),e(b,_,_,_,_))=>
    Int.compare(a,b))
                                   (initiate t_sen))

val () = writeToFile tex "entries.tex"
```

Appendix G

Querying ontologies with Prolog – source code

Below the full Prolog code is presented for our "pancreas knowledge base", together with querying facilities.

```
%% Explicit information from pancreas diagram:

db_class(cell).

db_class(nervous_system).

db_class(neuronal_schwann_cell).
db_r(neuronal_schwann_cell,isa,cell).
db_r(neuronal_schwann_cell,partof,nervous_system).

db_class(pancreas).

db_class(stem_cell).
db_r(stem_cell,isa,cell).

db_class(exocrine_cell).
db_r(exocrine_cell,isa,cell).
db_r(exocrine_cell,secrete,enzyme).
```

```
db_class(endocrine_cell).
db_r(endocrine_cell,isa,cell).
db_r(endocrine_cell,secrete,hormone).

db_class(pancreatic_cell).
db_r(pancreatic_cell,isa,cell).
db_r(pancreatic_cell,located_in,pancreas).

db_class(exocrine_pancreas).
db_r(exocrine_pancreas,partof,pancreas).

db_class(endocrine_pancreas).
db_r(endocrine_pancreas,partof,pancreas).

db_class(adult_stem_cell).
db_r(adult_stem_cell,isa,adult_stem_cell).

db_class(embryonic_stem_cell).
db_r(embryonic_stem_cell,isa,stem_cell).

db_class(islet_of_langerhans).
db_r(islet_of_langerhans,partof,endocrine_pancreas).

db_class(exocrine_pancreatic_cell).
db_r(exocrine_pancreatic_cell,isa,exocrine_cell).
db_r(exocrine_pancreatic_cell,isa,pancreatic_cell).
db_r(exocrine_pancreatic_cell,partof,exocrine_pancreas).

db_class(duct).
db_r(duct,partof,exocrine_pancreas).

db_class(endocrine_pancreatic_cell).
db_r(endocrine_pancreatic_cell,isa,endocrine_cell).
db_r(endocrine_pancreatic_cell,isa,pancreatic_cell).

db_class(capillary).
db_r(capillary,partof,islet_of_langerhans).

db_class(centroacinar_cell).
```

```
db_r(centroacinar_cell,isa,exocrine_pancreatic_cell).
db_r(centroacinar_cell,secrete,digestive_enzyme).
db_r(centroacinar_cell,primary_signal,secretin).

db_class(acinar_cell).
db_r(acinar_cell,isa,exocrine_pancreatic_cell).
db_r(acinar_cell,secrete,bicarbonate_ion).
db_r(acinar_cell,primary_signal,cck).

db_class(alpha_cell).
db_r(alpha_cell,isa,endocrine_pancreatic_cell).
db_r(alpha_cell,partof,islet_of_langerhans).
db_r(alpha_cell,secrete,glucagon).

db_class(beta_cell).
db_r(beta_cell,isa,endocrine_pancreatic_cell).
db_r(beta_cell,partof,islet_of_langerhans).
db_r(beta_cell,secrete,insulin).

db_class(gamma_cell).
db_r(gamma_cell,isa,endocrine_pancreatic_cell).
db_r(gamma_cell,partof,islet_of_langerhans).
db_r(gamma_cell,secrete,gastrin).
db_r(gamma_cell,secrete,somatostatin).

db_class(pp_cell).
db_r(pp_cell,isa,endocrine_pancreatic_cell).
db_r(pp_cell,partof,islet_of_langerhans).
db_r(pp_cell,secrete,pancreatic_polypeptide).

db_class(glucagon).

db_class(insulin).

db_class(gastrin).

db_class(somatostatin).

db_class(pancreatic_polypeptide).
```

```
%% Added for fixing knowledge base errors:

db_class(enzyme).
db_class(hormone).
db_class(digestive_enzyme).
db_class(secretin).
db_class(bicarbonate_ion).
db_class(cck).

%% Inference definitions.

db_relation(isa).
db_relation(partof).
db_relation(secrete).
db_relation(located_in).
db_relation(primary_signal).

inherited(partof).
inherited(secrete).
inherited(located_in).
inherited(primary_signal).

r_reflexive(X,_,X).

r_tr(X,R,Z) :- nonvar(Z) , r_tr1(X,R,Z).
r_tr(X,R,Z) :- nonvar(X) , r_tr2(X,R,Z).

r_tr1(X,R,Z) :- db_r(Y,R,Z) , db_r(X,R,Y).
r_tr1(X,R,Z) :- db_r(Y,R,Z) , r_tr1(X,R,Y).

r_tr2(X,R,Z) :- db_r(X,R,Y) , db_r(Y,R,Z).
r_tr2(X,R,Z) :- db_r(X,R,Y) , r_tr2(Y,R,Z).

reflexive(isa).
transitive(isa).
transitive(partof).

r(X,R,Y) :- db_r(X,R,Y).
```

```
r(X,R,Y) :- reflexive(R) , r_reflexive(X,R,Y).
r(X,R,Y) :- transitive(R) , r_tr(X,R,Y).

r(X,R,Y) :- inherited(R) , r(X,isa,Z) , diff(X,Z) , r(Z,R,Y).

%% Constraints:

error(undefined_relation(R)) :- r(_,R,_), \+ db_relation(R).
error(undefined_class(X)) :- r(X,_,_), \+ db_class(X).
error(undefined_class(X)) :- r(_,_,X), \+ db_class(X).

error(both_class_and_instance(X)) :- db_relation(X), db_class(X).

%% Querying:

ask([exist,Class1,which,Relation,Class2],[yes,for,instance,X,
    Relation,Y]):-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        r(X,isa,Class1),
        r(Y,isa,Class2),
        r(X,Relation,Y).

ask([every,Class1,Relation,Class2],[yes,every]):-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        \+ counterexample(Class1,Relation,Class2,_).

ask([every,Class1,Relation,Class2],E):-
        db_class(Class1),
        db_relation(Relation),
        db_class(Class2),
        counterexample(Class1,Relation,Class2,E).

counterexample(Class1,Relation,Class2,[no,because,X,not,Relation,
    Class2]):-
        r(X,isa,Class1),diff(X,Class1),
```

```
            not_r(X, Relation , Class2).

not_r(X,R,Y):- \+ r(X,R,Y).

diff(X,Y) :- \+ eq(X,Y).
eq(X,X).
```

Appendix H

Open–source licensing of the ontograbber

The following is a discussion on the topic of possibility of releasing the ontograbber under an open–source license.

The Ontograbber is an algorithm which associates ontological semantics with sentences. The retrieved ontological information takes the form of nodes in the generative ontology represented by so–called 'concept covers'. The result of the algoritmic analysis of a sentence is a collection of ontoterms.

H.1 What is free software?

The Free Software Foundation has defined what a free software is:

"Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it means that the program's users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).

- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this."

H.2 Why is Ontograbber a good candidate for becoming a free software?

The software developed in a academic environment, e.g. during a PhD studies has some unique characteristics as compared to a commercially developed software. The internal workings of the software are usually a public knowledge, described in detail in scientific papers and the thesis itself. Companies, on the other hand try to hide as much information as possible concerning the internal workings of the software they develop. They usually claim that this is to protect their trade secrets. In reality many companies do not allow people to look into the source code of their products due to poor programming practices and quality of the underlying design.

Since there is usually nothing to hide with academic software, this paper explores the possibility of releasing such a software as free software.

H.3 Community benefits

The free software enjoys a large community of people who use it. Those people for historical reasons have mostly very high computer skills. Many of them get involved in the development of the software and its environment. The ways people get involved is usually one of the following:

- They use the software, and if they encounter a bug, they report it. Since these are usually highly skilled users, the bug reports tend to be very detailed and of good quality. In return for reporting the bug, people can get a fix for it.

- They write documentation. This can e.g. take the form of a WIKI, where users themselves document the software. The motivation for doing that might be: "You (developers) better concentrate on developing the software, while we (users) will document it for you." People realize that if they take the burden of writing the documentation off the developers, the software will be developed more quickly.

- They discuss and support. It is much easier for users to get support for a free software product than for a closed alternative. This is because there is a lot of other users of the software that are ready to share their knowledge and help when there are problems. This can take the form of a mailing list, a forum website, Internet Relay Chat, or a local user group. Help is usually provided promptly, sometimes in minutes after a question is asked. At the Technical University of Denmark we have a "Linux User Group", which meets at regular intervals.

- They develop the software. People with programming skills often will offer to fix a bug themselves. They might send a patch together with a bug report. There is also a community of "bug squashers", who follow the bug reports and donate their free time by fixing known issues.

H.4 Why do free software communities form?

The reasons people get involved in free software communities are the following:

- They get the software for free, based on hard work of others. They want to contribute back in return.

- They want to learn. Getting involved in a open source project is a great way of learning or mastering programming skills. One can also learn how large software projects are managed successfully, e.g. those involving thousands of programmers.

- They make money on the product. E.g. a company that uses the Linux operating system kernel as part of their commercial product might need to extend it in some way in order to use it successfully in their product. They may e.g. write a new device driver for hardware that they have developed.

- They want to be a part of a friendly community. People are social beings. They need to belong to some communities. People with high computer skills are very likely to feel well in a free software community.

H.5 Main types of open source software licenses

1. Freedom–preserving licenses. These licenses preserve the basic liberties of the free software. This means that the receipient of the software covered

by such a license cannot in any way turn it into non–free software. The most commonly used license in this group is the GNU General Public License.

2. Permissive licenses. These licenses allow the receipient of a open source software to turn it into non-free software or even a completely closed software and distribute it further as such. Why would anybody need that? This commonly occurs when some company wants to use some open source software product or library as part of their commercial product, without the combining product becoming an open source product. The most common licences of this group are GNU Lesser General Public License and BSD software license.

H.6 Business models for free software and open source software

H.6.1 Donations

It is very common for free software to accept donations from users who would like to support the project or the developers financially.

H.6.2 Double licensing

Many companies currently involved in the free software business make money on the so–called double licensing. This is a business model where exactly the same product is provided as both a free–licensed one and as a closed–licensed one.

The free–licensed product is usually given away for free to anybody who wants it. This helps building a large community of users, wide–spreading and advertising the software. The developers can also get a lot back from those users, e.g. free documentation, free bug reports, or even fixes for them, and free support for other users of this software.

The closed–licensed product is sold to customers who need a closed license for some reason.

H.6.3 Support

Many companies developing free software are giving copies of the software and its source code for free, while charging users for optional support with respect

to the software. Many customers, including most corporate customers need a guarantee of professional support when they decide to use a product. Free software vendors may be selling time–limited access to such a support. Technicians providing such a support may solve it using on–line communication, telephone, or by visiting the customer.

H.6.4 Extra information

Many free software developers make money not on the product itself, but on the information about it. It is quite common for a developer to write a book describing a particular free software. Information vending can also take form of consulting, conducting courses or workshops, etc.

H.6.5 Bounties

Some free software developers propose to add a requested extension to an existing product for some financial compensation, called a bounty. If a user is in need of a particular fix/extension, she may consider paying such a bounty so that the developers will solve her problems quickly.

H.7 Conclusion

We believe that the Ontograbber is well–fit for becoming a free software. Even though it would be released to any interested party for free, there still exist business models that allow the developing team to generate revenue.

Appendix I

Tests of the ontograbber algorithm on additional 27 sentences

Below we present the results of testing the final ontograbber algorithm on additional 27 sentences. Fo each sentence the lexicon and ontology had to be extended manually, in order to give the algorithm a chance of coping with the given sentence. Unfortunately for these sentences we were unable to obtain a golden standard from experts against which we could compare the results.

The input sentence 11:

```
molecular cloning and characterization of an insulin -
    regulatable glucose transporter
```

Ontograbber's tagged sentence:

```
molecular[A] cloning[N] and[] characterization[N] of[] an
    [] insulin[N] -[] regulatable[A] glucose[N]
    transporter[N]
```

Ontograbber's output ontosemantics:

- (0,2,NP,cloning$\left[\textit{chr}\text{: molecular}\right]$,"molecular cloning and characterization")
- (0,2,NP,characterization,"molecular cloning and characterization")
- (3,3,NP,insulin,"insulin")
- (4,4,A,regulatable,"regulatable")
- (5,6,NP,transporter$\left[\textit{wrt}\text{: glucose}\right]$,"glucose transporter")

The input sentence 12:

```
glucokinase is the likely mediator of glucosensing in
    both glucose - excited and glucose - inhibited central
     neurons
```

Ontograbber's tagged sentence:

```
glucokinase[N] is[IS] the[] likely[] mediator[N] of[]
    glucosensing[N] in[] both[] glucose[N] -[] excited[VBD
     VBN] and[] glucose[N] -[] inhibited[VBD VBN] central[
    A] neurons[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,glucokinase,"glucokinase")
- (1,1,IS,is,"is")
- (2,2,NP,mediator,"mediator")
- (3,3,NP,glucosensing,"glucosensing")
- (4,9,NP,neuron$\begin{bmatrix}\textit{chr}\text{: central}\\ \textit{pntof}\text{: inhibition}\left[\textit{agt}\text{: glucose}\right]\end{bmatrix}$,"glucose - excited and glucose - inhibited central neurons")
- (4,9,NP,neuron$\begin{bmatrix}\textit{chr}\text{: central}\\ \textit{pntof}\text{: inhibition}\begin{bmatrix}\textit{agt}\text{: glucose}\\ \textit{agt}\text{: glucose}\end{bmatrix}\end{bmatrix}$,"glucose - excited and glucose - inhibited central neurons")

- (4,9,NP,neuron $\begin{bmatrix} chr\text{: central} \\ pntof\text{: excitement} \begin{bmatrix} agt\text{: glucose} \end{bmatrix} \end{bmatrix}$,"glucose - excited and glucose - inhibited central neurons")

The input sentence 13:

```
preferably , the insulin secretion is glucose -
    stimulated insulin secretion
```

Ontograbber's tagged sentence:

```
preferably [] ,[,] the [] insulin [N] secretion [N] is [IS]
    glucose [N] -[] stimulated [VBD VBN] insulin [N]
    secretion [N]
```

Ontograbber's output ontosemantics:

- (0,0,,,nil,",")
- (1,5,PSV,stimulation $\begin{bmatrix} agt\text{: glucose} \\ pnt\text{: secretion} \begin{bmatrix} pnt\text{: insulin} \end{bmatrix} \end{bmatrix}$," insulin secretion is glucose - stimulated")
- (3,7,PSV,stimulation $\begin{bmatrix} agt\text{: glucose} \\ pnt\text{: secretion} \begin{bmatrix} pnt\text{: insulin} \end{bmatrix} \end{bmatrix}$," insulin secretion is glucose - stimulated")

The input sentence 14:

```
the entry of glucose into the adipocyte requires the
    activity of the insulin sensitive glucose transporters
```

Ontograbber's tagged sentence:

```
the [] entry [N] of [] glucose [N] into [] the [] adipocyte [N]
    requires [TV] the [] activity [N] of [] the [] insulin [N]
    sensitive [A] glucose [N] transporters [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,entery,“ entry”)
- (1,1,NP,glucose,“glucose”)
- (2,2,NP,adipocyte,“ adipocyte”)
- (3,3,TV,requirement,“requires”)
- (4,4,NP,activity,“ activity”)
- (5,5,NP,insulin,“ insulin”)
- (6,6,A,sensitive,“sensitive”)
- (7,8,NP,transporter[*wrt*: glucose],“glucose transporters”)

The input sentence 15:

```
however , insulin also acts to inhibit the activity of
    glucose-6-phosphatase
```

Ontograbber's tagged sentence:

```
however [] ,[ ,] insulin [N] also [] acts [TV] to [] inhibit [TV
    ] the [] activity [N] of [] glucose-6-phosphatase [N]
```

Ontograbber's output ontosemantics:

- (0,0,,,nil,“,”)
- (1,1,NP,insulin,“insulin”)
- (2,2,TV,acting,“acts”)
- (3,4,VP,inhibition[*pnt*: activity],“inhibit activity”)
- (5,5,NP,glucose-6-phosphatase,“glucose-6-phosphatase”)

The input sentence 16:

```
increased glucose concentrations are sensed upon
    transport across the plasma membrane by glucose
    transporter glycoprotein resulting in insulin
    secretion
```

Ontograbber's tagged sentence:

```
increased[VBD VBN] glucose[N] concentrations[N] are[IS]
    sensed[VBD VBN] upon[] transport[N TV] across[] the[]
    plasma[N] membrane[N] by[] glucose[N] transporter[N]
    glycoprotein[N] resulting[VBG] in[] insulin[N]
    secretion[N]
```

Ontograbber's output ontosemantics:

- (0,0,VBD,increase,"increased")
- (0,0,VBN,increased,"increased")
- (1,2,NP,concentration[*wrt*: glucose],"glucose concentrations")
- (3,4,PSV,sensing,"are sensed")
- (5,5,TV,transport,"transport")
- (5,5,NP,transport,"transport")
- (6,6,NP,plasma," plasma")
- (7,7,NP,membrane,"membrane")
- (8,9,NP,transporter[*wrt*: glucose],"glucose transporter")
- (10,10,NP,glycoprotein,"glycoprotein")
- (11,11,VBG,result,"resulting")
- (12,13,NP,secretion[*pnt*: insulin],"insulin secretion")

The input sentence 17:

```
these tissues require insulin for facilitated transport
    of glucose and express the insulin - responsive
    transporter glut4
```

Ontograbber's tagged sentence:

```
these[] tissues[N] require[TV] insulin[N] for[]
    facilitated[VBD VBN] transport[N TV] of[] glucose[N]
    and[] express[TV] the[] insulin[N] -[] responsive[A]
    transporter[N] glut4[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,tissue,"tissues")
- (1,2,VP,requirement$\left[\textit{pnt}\text{: insulin}\right]$,"require insulin")
- (3,5,NP,transport$\begin{bmatrix}\textit{pnt}\text{: glucose}\\ \textit{pntof}\text{: facilitation}\end{bmatrix}$,"facilitated transport of glucose")
- (6,6,TV,expression,"express")
- (7,7,NP,insulin," insulin")
- (8,8,A,responsive,"responsive")
- (9,10,NP,glut4$\left[\textit{wrt}\text{: transporter}\right]$,"transporter glut4")

The input sentence 18:

```
in the liver , insulin stimulates glucose incorporation
    into glycogen and inhibits the production of glucose
```

Ontograbber's tagged sentence:

```
in[] the[] liver[N] ,[,] insulin[N] stimulates[TV]
    glucose[N] incorporation[N] into[] glycogen[N] and[]
    inhibits[TV] the[] production[N] of[] glucose[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,liver,“ liver”)
- (1,1,,,nil,“,”)
- (2,5,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: incorporation}\left[pnt\text{: glucose}\right] \end{bmatrix}$,“insulin stimulates glucose incorporation”)
- (6,9,S,inhibition $\begin{bmatrix} agt\text{: glycogen} \\ pnt\text{: production}\left[pnt\text{: glucose}\right] \end{bmatrix}$,“glycogen inhibits production of glucose”)

The input sentence 19:

```
the effect of insulin was the opposite of glucose
```

Ontograbber's tagged sentence:

```
the[] effect[N] of[] insulin[N] was[IS] the[] opposite[]
    of[] glucose[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,effect,“ effect”)
- (1,1,NP,insulin,“insulin”)
- (2,2,IS,is,“was”)
- (3,3,NP,glucose,“glucose”)

The input sentence 20:

```
insulin regulation of the two glucose transporters in 3t3
    -l1 adipocytes
```

Ontograbber's tagged sentence:

```
insulin [N] regulation [N] of [] the [] two [] glucose [N]
    transporters [N] in [] 3t3-l1 [N] adipocytes [N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,regulation[*agt*: insulin],"insulin regulation")
- (2,3,NP,transporter[*wrt*: glucose],"glucose transporters")
- (4,5,NP,adipocyte[*pof*: q-3t3l1],"3t3-l1 adipocytes")
- (4,5,NP,adipocyte[*loc*: q-3t3l1],"3t3-l1 adipocytes")

The input sentence 21:

```
involved in insulin regulated translocation of the
    glucose transporter glut4
```

Ontograbber's tagged sentence:

```
involved [VBD VBN] in [] insulin [N] regulated [VBD VBN]
    translocation [N] of [] the [] glucose [N] transporter [N]
    glut4 [N]
```

Ontograbber's output ontosemantics:

- (0,0,VBD,involvment,"involved")
- (0,0,VBN,involvment,"involved")

- (1,1,NP,insulin,"insulin")
- (2,6,NP,translocation $\begin{bmatrix} pnt\text{: glut4}\left[wrt\text{: transporter}\left[wrt\text{: glucose}\right]\right] \\ pntof\text{: regulation} \end{bmatrix}$,"regulated translocation of glucose transporter glut4")
- (2,6,NP,translocation $\begin{bmatrix} pnt\text{: glut4}\begin{bmatrix} wrt\text{: glucose} \\ wrt\text{: transporter} \end{bmatrix} \\ pntof\text{: regulation} \end{bmatrix}$,"regulated translocation of glucose transporter glut4")

The input sentence 22:

```
the glucose level was detected with a glucose detection
    kit
```

Ontograbber's tagged sentence:

```
the [] glucose [N] level [N] was [IS] detected [VBN] with [] a
    [] glucose [N] detection [N] kit [N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,level$\left[wrt\text{: glucose}\right]$," glucose level")
- (2,3,PSV,detection,"was detected")
- (4,4,NP,glucose," glucose")
- (5,5,NP,detection,"detection")
- (6,6,NP,kit,"kit")

The input sentence 23:

```
cellular manifestations of insulin resistance include
    impaired insulin - stimulated glucose uptake by
    peripheral tissues and impaired glucose disposal
```

Ontograbber's tagged sentence:

```
cellular[A] manifestations[N] of[] insulin[N] resistance[
    N] include[TV] impaired[VBD VBN] insulin[N] -[]
    stimulated[VBD VBN] glucose[N] uptake[N] by[]
    peripheral[A] tissues[N] and[] impaired[VBD VBN]
    glucose[N] disposal[N]
```

Ontograbber's output ontosemantics:

- (0,0,A,cellular,"cellular")
- (1,1,NP,manifestation,"manifestations")
- (2,2,NP,insulin,"insulin")
- (3,3,NP,resistance,"resistance")
- (4,4,TV,inclusion,"include")
- (5,5,VBD,impairment,"impaired")
- (5,5,VBN,impairment,"impaired")
- (6,9,NP,uptake$\begin{bmatrix} pnt\text{: glucose} \\ pntof\text{: stimulation}\big[agt\text{: insulin}\big] \end{bmatrix}$,"insulin - stimulated glucose uptake")
- (10,10,A,peripheral,"peripheral")
- (11,11,NP,tissue,"tissues")
- (12,12,VBD,impairment,"impaired")
- (12,12,VBN,impairment,"impaired")
- (13,14,NP,disposal$\big[pnt\text{: glucose}\big]$,"glucose disposal")

The input sentence 24:

```
high glucose levels promote insulin resistance , and
    insulin resistance generates prolonged elevations of
    serum glucose concentration
```

Ontograbber's tagged sentence:

```
high [A] glucose [N] levels [N] promote [TV] insulin [N]
    resistance [N] , [ , ] and [] insulin [N] resistance [N]
    generates [TV] prolonged [VBN] elevations [] of [] serum [N
    ] glucose [N] concentration [N]
```

Ontograbber's output ontosemantics:

- (0,2,NP,level $\begin{bmatrix} chr\text{: high} \\ wrt\text{: glucose} \end{bmatrix}$, "high glucose levels")
- (3,3,TV,promotion, "promote")
- (4,4,NP,insulin, "insulin")
- (5,5,NP,resistance, "resistance")
- (6,6,,,nil, ",")
- (7,7,NP,insulin, "insulin")
- (8,8,NP,resistance, "resistance")
- (9,9,TV,generation, "generates")
- (10,10,VBN,prolongation, "prolonged")
- (11,11,NP,serum, "serum")
- (12,13,NP,concentration $\left[wrt\text{: glucose}\right]$, "glucose concentration")

The input sentence 25:

```
in the liver , insulin inhibits the release of glucose
    from glycogen and the synthesis of new glucose
```

Ontograbber's tagged sentence:

```
in [] the [] liver [N] ,[,] insulin [N] inhibits [TV] the []
    release [N] of [] glucose [N] from [] glycogen [N] and []
    the [] synthesis [N] of [] new [] glucose [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,liver,“ liver”)
- (1,1,,,nil,“,”)
- (2,5,S,inhibition $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: release} \begin{bmatrix} pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$,“insulin inhibits release of glucose”)
- (6,7,NP,glycogen,“glycogen and synthesis”)
- (6,7,NP,synthesis,“glycogen and synthesis”)
- (8,8,NP,glucose,“glucose”)

The input sentence 26:

```
insulin lowers blood glucose levels and promotes
    transport and entry of glucose into muscle cells and
    other tissues
```

Ontograbber's tagged sentence:

```
insulin [N] lowers [TV] blood [N] glucose [N] levels [N] and []
    promotes [TV] transport [N TV] and [] entry [N] of []
    glucose [N] into [] muscle [N] cells [N] and [] other []
    tissues [N]
```

Ontograbber's output ontosemantics:

- (0,2,S,decrease $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: blood} \end{bmatrix}$,“insulin lowers blood”)

- (2,8,S,promotion $\begin{bmatrix} agt\text{: level}\left[wrt\text{: glucose}\left[src\text{: blood}\right]\right] \\ pnt\text{: transport}\left[pnt\text{: glucose}\right] \end{bmatrix}$, "blood glucose levels promotes transport and entry of glucose")
- (2,8,S,promotion $\begin{bmatrix} agt\text{: level}\begin{bmatrix} loc\text{: blood} \\ wrt\text{: glucose} \end{bmatrix} \\ pnt\text{: transport}\left[pnt\text{: glucose}\right] \end{bmatrix}$, "blood glucose levels promotes transport and entry of glucose")
- (9,10,NP,cell$\left[pof\text{: muscle}\right]$, "muscle cells")
- (9,10,NP,cell$\left[loc\text{: muscle}\right]$, "muscle cells")
- (11,11,NP,tissue, "tissues")

The input sentence 27:

```
in the liver , insulin inhibits the release of glucose
    from glycogen
```

Ontograbber's tagged sentence:

```
in [] the [] liver [N] ,[ ,] insulin [N] inhibits [TV] the []
    release [N] of [] glucose [N] from [] glycogen [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,liver, " liver")
- (1,1,,,nil, ",")
- (2,5,S,inhibition $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: release}\left[pnt\text{: glucose}\right] \end{bmatrix}$, "insulin inhibits release of glucose")
- (6,6,NP,glycogen, "glycogen")

The input sentence 28:

```
addition of name to the incubation mixture inhibited the
    insulin - mimicking effect of allimin on both glucose
    transport and glucose oxidation
```

Ontograbber's tagged sentence:

```
addition[N] of[] name[] to[] the[] incubation[N] mixture[
    N] inhibited[VBD VBN] the[] insulin[N] -[] mimicking[]
     effect[N] of[] allimin[N] on[] both[] glucose[N]
    transport[N TV] and[] glucose[N] oxidation[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,addition,"addition")
- (1,1,NP,incubation," incubation")
- (2,2,NP,mixture,"mixture")
- (3,3,VBD,inhibition,"inhibited")
- (3,3,VBN,inhibition,"inhibited")
- (4,4,NP,insulin," insulin")
- (5,5,NP,effect,"effect")
- (6,6,NP,allimin,"allimin")
- (7,10,NP,transport[*pnt*: glucose],"glucose transport and glucose oxidation")
- (7,10,NP,oxidation[*pnt*: glucose],"glucose transport and glucose oxidation")

The input sentence 29:

```
the glucose formed is then transported by the glucose
    carrier of the organism
```

Ontograbber's tagged sentence:

```
the [] glucose [N] formed [VBD VBN] is [IS] then []
    transported [VBD VBN] by [] the [] glucose [N] carrier [N]
    of [] the [] organism [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,glucose,“ glucose”)
- (1,1,VBD,forming,“formed”)
- (1,1,VBN,forming,“formed”)
- (2,3,PSV,transport,“is transported”)
- (4,5,NP,carrier[*wrt*: glucose],“ glucose carrier”)
- (6,6,NP,organism,“ organism”)

The input sentence 30:

```
a smaller auc for glucose ( smaller blood glucose
    excursion after glucose challenge ) indicates better
    glucose tolerance
```

Ontograbber's tagged sentence:

```
a [] smaller [A] auc [N] for [] glucose [N] ( [] smaller [A]
    blood [N] glucose [N] excursion [N] after [] glucose [N]
    challenge [N] ) [] indicates [TV] better [] glucose [N]
    tolerance [N]
```

Ontograbber's output ontosemantics:

- (0,0,A,smaller,“smaller”)
- (1,1,NP,auc,“auc”)
- (2,2,NP,glucose,“glucose”)

- (3,3,A,smaller,"smaller")
- (4,6,NP,excursion$\left[\textit{pnt}\text{: glucose}\left[\textit{src}\text{: blood}\right]\right]$,"blood glucose excursion")
- (4,6,NP,excursion$\begin{bmatrix}\textit{loc}\text{: blood}\\ \textit{pnt}\text{: glucose}\end{bmatrix}$,"blood glucose excursion")
- (6,7,NP,excursion$\left[\textit{pnt}\text{: glucose}\right]$,"glucose excursion")
- (8,8,NP,challenge,"challenge")
- (9,9,TV,indication,"indicates")
- (10,11,NP,tolerance$\left[\textit{pnt}\text{: glucose}\right]$,"glucose tolerance")

The input sentence 31:

```
general caspases inhibitor ( z-asp-ch2-dcb ) inhibited
    cell death
```

Ontograbber's tagged sentence:

```
general[] caspases[N] inhibitor[N] ([] z-asp-ch2-dcb[] )
    [] inhibited[VBD VBN] cell[N] death[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,caspase,"caspases")
- (1,1,NP,inhibitor,"inhibitor")
- (2,4,NP,death$\begin{bmatrix}\textit{loc}\text{: cell}\\ \textit{pntof}\text{: inhibition}\end{bmatrix}$,"inhibited cell death")

The input sentence 32:

```
glucose tolerance tends to progressively decline with age
```

Ontograbber's tagged sentence:

```
glucose [N] tolerance [N] tends [] to [] progressively []
    decline [TV] with [] age [N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,tolerance$\left[\textit{pnt}\text{: glucose}\right]$,"glucose tolerance")
- (2,2,TV,decline,"decline")
- (3,3,NP,age,"age")

The input sentence 33:

```
glucose - dependent insulinotropic polypeptide ( gip )
    potentiates glucose - induced insulin secretion
```

Ontograbber's tagged sentence:

```
glucose [N] -[] dependent [A] insulinotropic [A] polypeptide
    [N] ( [] gip [] ) [] potentiates [TV] glucose [N] -[]
    induced [VBD VBN] insulin [N] secretion [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,glucose,"glucose")
- (1,1,A,dependent,"dependent")
- (2,8,S,potentiation $\begin{bmatrix} \textit{agt}\text{: polypeptide}\left[\textit{chr}\text{: insulinotropic}\right] \\ \textit{pnt}\text{: secretion}\begin{bmatrix}\textit{pnt}\text{: insulin} \\ \textit{pntof}\text{: induction}\left[\textit{agt}\text{: glucose}\right]\end{bmatrix}\end{bmatrix}$,"insulinotropic polypeptide potentiates glucose - induced insulin secretion")

The input sentence 34:

```
background insulin stimulates glucose transport in muscle
    and fat
```

Ontograbber's tagged sentence:

```
background[N] insulin[N] stimulates[TV] glucose[N]
   transport[N TV] in[] muscle[N] and[] fat[N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,background,"background")
- (1,6,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: fat} \end{bmatrix}$,"insulin stimulates glucose transport in muscle and fat")
- (1,6,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: transport} \begin{bmatrix} loc\text{: fat} \\ pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$,"insulin stimulates glucose transport in muscle and fat")
- (1,6,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ pnt\text{: transport} \begin{bmatrix} loc\text{: muscle} \\ pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$,"insulin stimulates glucose transport in muscle and fat")
- (1,6,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ loc\text{: muscle} \\ pnt\text{: transport} \begin{bmatrix} pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$,"insulin stimulates glucose transport in muscle and fat")
- (1,6,S,stimulation $\begin{bmatrix} agt\text{: insulin} \\ loc\text{: fat} \\ pnt\text{: transport} \begin{bmatrix} pnt\text{: glucose} \end{bmatrix} \end{bmatrix}$,"insulin stimulates glucose transport in muscle and fat")

The input sentence 35:

```
most of the compounds which inhibit tyrosine
    phosphorylation of the insulin receptor also inhibited
     glucose uptake in the same cells
```

Ontograbber's tagged sentence:

```
most [] of [] the [] compounds [N] which [] inhibit [TV]
    tyrosine [N] phosphorylation [N] of [] the [] insulin [N]
    receptor [N] also [] inhibited [VBD VBN] glucose [N]
    uptake [N] in [] the [] same [] cells [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,compound,“ compounds”)
- (1,1,TV,inhibition,“inhibit”)
- (2,2,NP,tyrosine,“tyrosine”)
- (3,3,NP,phosphorylation,“phosphorylation”)
- (4,4,NP,insulin,“ insulin”)
- (5,5,NP,receptor,“receptor”)
- (6,8,NP,uptake $\begin{bmatrix} pnt\text{: glucose} \\ pntof\text{: inhibition} \end{bmatrix}$,“inhibited glucose uptake”)
- (9,9,NP,cell,“cells”)

The input sentence 36:

```
inhibition of zcytor17lig activity could thus inhibit
    growth of such cells
```

Ontograbber's tagged sentence:

```
inhibition [N] of [] zcytor17lig [N] activity [N] could []
    thus [] inhibit [TV] growth [N] of [] such [] cells [N]
```

Ontograbber's output ontosemantics:

- (0,0,NP,inhibition,"inhibition")
- (1,1,NP,zcytor17lig,"zcytor17lig")
- (2,2,NP,activity,"activity")
- (3,4,VP,inhibition$\left[\textit{pnt}\text{: growth}\right]$,"inhibit growth")
- (5,5,NP,cell,"cells")

The input sentence 37:

```
the transport of glucose across a cellular membrane is
    stimulated by insulin binding to its insulin receptor
    as part of an insulin signalling pathway
```

Ontograbber's tagged sentence:

```
the [] transport [N TV] of [] glucose [N] across [] a []
    cellular [A] membrane [N] is [IS] stimulated [VBD VBN] by
    [] insulin [N] binding [N VBG] to [] its [] insulin [N]
    receptor [N] as [] part [] of [] an [] insulin [N]
    signalling [] pathway [N]
```

Ontograbber's output ontosemantics:

- (0,1,NP,transport$\left[\textit{pnt}\text{: glucose}\right]$," transport of glucose")
- (2,2,A,cellular,"cellular")
- (3,6,PSV,stimulation$\begin{bmatrix}\textit{agt}\text{: insulin}\\ \textit{pnt}\text{: membrane}\end{bmatrix}$,"membrane is stimulated by insulin")
- (7,7,VBG,binding,"binding")
- (7,7,NP,binding,"binding")

- (8,8,NP,insulin,“ insulin”)
- (9,9,NP,receptor,“receptor”)
- (10,10,NP,insulin,“ insulin”)
- (11,11,NP,pathway,“pathway”)

Bibliography

[1] Troels Andreasen and Jørgen Fischer Nilsson. Grammatical specification of domain ontologies. *Data Knowl. Eng.*, 48(2):221–230, 2004.

[2] P. Andrews. Classical type theory, 2001. Peter Andrews. Classical type theory. In Alan Robinson and Andrei Voronkov, editors, Handbook of Automated Reasoning,volume 2, chapter 15, pages 965-1007. North-Holland, 2001. 43.

[3] Peter B. Andrews. *An introduction to mathematical logic and type theory: to truth through proof*. Academic Press Professional, Inc., San Diego, CA, USA, 1986.

[4] John Aycock and Nigel Horspool. Practical earley parsing, 2002.

[5] F. Baader and W. Nutt. Basic description logics, 2003.

[6] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics*, pages 86–90, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[7] Collin F. Baker and Hiroaki Sato. The framenet data and software. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 161–164, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[8] M. Ben-Ari. *Mathematical logic for computer science*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[9] Gilad Ben-Avi and Nissim Francez. Categorial grammar with ontology-refined types. In *Proceedings of CG04*, 2004.

[10] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI Publications, 2005.

[11] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[12] Chris Brink, Katarina Britz, and Renate A. Schmidt. Peirce algebras. *Formal Aspects of Computing*, 6(3):339–358, April 1994. Also available as Research Report MPI-I-92-229, Max-Planck-Institut fr Informatik, Saarbrcken, Germany (July 1992), and as Research Report RR 140, Department of Mathematics, University of Cape Town, Cape Town, South Africa (August 1992). An extended abstract appears in Nivat, M., Rattray, C., Rus, T. and Scollo, G. (eds) , *Algebraic Methodology and Software Technology (AMAST'93): Proceedings of the 3rd International Conference on Algebraic Methodology and Software Technology. Workshops in Computing* Series, Springer-Verlag, London, 165-168 (1994).

[13] Bob Carpenter. *Type-logical semantics*. MIT Press, Cambridge, MA, USA, 1998.

[14] Wikipedia contributors. Alexius meinong. *http://en.wikipedia.org/wiki/Alexius_Meinong*.

[15] Wikipedia contributors. Frame semantics. *http://en.wikipedia.org/wiki/Frame_semantics_(linguistics)*.

[16] Andrew Dolbey, Michael Ellsworth, and Jan Scheffczyk. Bioframenet: A domain-specific framenet extension with links to biomedical ontologies. In Olivier Bodenreider, editor, *KR-MED*, volume 222 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[17] Chris Fox. *The Ontology of Language: properties, individuals and discourse*. Lecture Notes of The Center for the Study of Language and Information (CSLI). The Center for the Study of Language and Information (CSLI), 2000.

[18] L. T. F. Gamut. *Logic, Language and Meaning*, volume 1. University of Chicago Press, 1991.

[19] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. The MIT Press, Cambridge, Massachusetts, 2000.

[20] Peter Gerstl and Simone Pribbeenow. Midwinters, end games, and body parts: a classification of part–whole relation. In *Human-Computer Studies*, pages 865–889, 1995.

[21] Peter Gerstl and Simone Pribbenow. Midwinters, end games, and body parts: a classification of part-whole relations. *Int. J. Hum.-Comput. Stud.*, 43(5-6):865–889, 1995.

[22] Roxana Girju, Dan Moldovan, Marta Tatu, and Daniel Antohe. On the semantics of noun compounds. *Comput. Speech Lang.*, 19(4):479–496, 2005.

[23] Asuncion Gomez-Perez, Oscar Corcho, and Mariano Fernandez-Lopez. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition (Advanced Information and Knowledge Processing)*. Springer, July 2004.

[24] P. A. Jensen and J. F. Nilsson. Ontology-based semantics for prepositions. In *Syntax and Semantics of Prepositions*, Text, Speech and Language Technology, Vol. 29. Springer, 2006.

[25] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition, 2008.

[26] Edward L. Keenan and Leonard M. Faltz. *Boolean Semantics for Natural Language*. Reidel., Dordrecht, 1986.

[27] Douglas Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[28] Godehard Link. Algebraic semantics for natural language: some philosophy, some application. *Int. J. Hum.-Comput. Stud.*, 43(5-6):765–784, 1995.

[29] Dave MacQueen. The standard ml of new jersey website. *http://www.smlnj.org/index.html*.

[30] Matt McGee. searchengineland.com. *http://searchengineland.com/by-the-numbers-twitter-vs-facebook-vs-google-buzz-36709*.

[31] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA, 1990.

[32] J. F. Nilsson. Ontological constitutions for classes and properties. In P. Øhrstrøm H. Schaerfe, P. Hitzler, editor, *14th Int. Conf. on Conceptual Structures, ICCS 2006*, Lecture Notes in Artificial Intelligence LNAI 4068. Springer, 2006.

[33] Sergei Nirenburg and Victor Raskin. *Ontological Semantics*. Language, Speech, and Communication. MIT Press, Cambridge, Mass., 2004.

[34] R. T. Oehrle, E. Bach, and D. Wheeler. *Categorial Grammars and Natural Language Structures*. Reidel, Dordrecht, 1988.

[35] F. Pereira and S. Shieber. *Prolog and natural-language analysis*. Stanford : Center for the Study of Language and Information, 1987.

[36] Benjamin C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

[37] The Mercury Project. The mercury project benchmarks. *http://www.mercury.cs.mu.oz.au/information/benchmarks.html*.

[38] Jørgen Fischer Nilsson and Nikolaj Oldager. Introduction to orders and lattices. 2002.

[39] Josef Ruppenhofer, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. FrameNet II: Extended theory and practice. Technical report, ICSI, 2005.

[40] Sag, T. Wasow, and E. Bender. *Syntactic Theory: a formal introduction, Second Edition*. 2003.

[41] B Smith and C Rosse. The role of foundational relations in the alignment of biomedical ontologies. In *MEDINFO 2004. Proceedings of the 11th World Congress on Medical Informatics; 2004 Sep 7-11*. IOS Press, 2004.

[42] Barry Smith. Beyond concepts: Ontology as reality representation.

[43] Holger Stenzhorn. Basic formal ontology website. *http://www.ifomis.uni-saarland.de/bfo/*.

[44] Author unknown. The mercury project introduction. *http://www.cs.mu.oz.au/research/mercury/*.

[45] Author unknown. Obo foundry website. *http://obofoundry.org/*.

[46] Jørgen Villadsen. *Nabla: A Linguistic System based on Type Theory*. Lit, 2010.

[47] Christopher A. Welty and David A. Ferrucci. What's in an instance?

[48] Dominic Widdows. *Geometry and Meaning*. CLSI Publications, Stanford, CA, USA, 2004.

[49] Mauhab Zareh. Framenet frequently asked questions. *http://framenet.icsi.berkeley.edu/*.