Semantic Search in Scientific Texts using Natural Logics

Master thesis

Author: Quentin Tresontani (s151409)
Supervisor: Jørgen Fischer Nilsson

Technical University of Denmark,
June 2017.

DTU Compute
Department of Applied Mathematics and Computer Science

Master of Science in Engineering
Computer Science and Engineering

Semantic Search in Scientific Texts using Natural Logics
Master thesis

Author: Quentin Tresontani (s151409)
Supervisor: Jørgen Fischer Nilsson
Author email: *s151409@student.dtu.dk*
Supervisor email: *jfni@dtu.dk*

Technical University of Denmark
Department of Applied Mathematics and Computer Science
DTU Compute
Richard Petersens Plads
Building 324
2800 Kgs. Lyngby
DENMARK

Tel. +45 4525 4800
www.compute.dtu.dk

Preface

This Master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a Master degree in Computer Science and Engineering.

Kongens Lyngby, Denmark, June 23, 2017

[illegible]

Quentin Tresontani

Abstract

This thesis reviews the use of Natural Logics as a solution to handle semantic search in scientific texts. Various frameworks have been proposed to deal with semantics such as FOL or DL, but their syntax always remains drastically different from natural language. This makes them hardly understandable by inexperienced users, but also rises the problem of an automated translation process from and to these languages.

This thesis tries to overcome this problem, by working on Natural Logics, a reasoning framework using natural language as vessel where reasoning can be conducted directly by using rules reflecting intuitive reasoning in natural language.

After defining a first approach with limited scope named Core Natural Logics (III), this thesis focuses on defining a set of extensions for Core Natural Logics in order to define a grammar and a parser able to cope with most linguistic structures of scientific texts (IV). The thesis especially focuses on defining a domain specific ontological analysis of the concepts in order to avoid syntactic ambiguities and to infer underlying relations between elements of the sentence (V).

An algorithm to decompose Natural Logics propositions into a concept graph representation integrated in the domain ontology is then proposed (VI). This enables Natural Logics to draw complex inferences proofs using graph search and to return the answer in natural language (VII). Finally the correctness of Natural Logics grammar, of its graph decomposition and graph search are assessed (VIII), before discussing Natural Logics current limitations, potential improvements and the related future work (IX).

It is hoped this thesis and the implemented Prolog prototype will demonstrate the reader the benefits of using Natural Logics for semantic search.

Acknowledgements

I would like to thank my supervisor Prof. Dr. Jørgen Fischer Nilsson for his availability and his patience in guiding me to complete this thesis project. I am grateful for all his constructive feedback discussions, his help to guide me stepwise through learning different semantic approaches and for the support he has given to me during this project.

I would also like to thank Troels Andreasen, Henrik Bulskov and Per Anker Jensen that I have met at Roskilde University to discuss Natural Logics with them. This meeting and the feedback I received were particularly interesting for this Master Thesis.

Last but not least, I thank my family, my friends and Laura for their continuous moral support to keep me going during my thesis.

DTU Compute, 2017

Quentin

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Today, more and more knowledge is accessible through the Internet. We usually use search engine in order to access it. Even if the major search engine companies started to show interest for semantic search, they do not process any advanced semantic analysis for now. Search engines answer most of the time with the best documents matching the keywords. It is then not granted that documents using synonyms will be considered. Similarly, the context or the intention of the user aren't considered. If the user has a specific question in mind, this question cannot be directly asked. He needs to pre-process his search by carefully choosing keywords that will put him on the track of the answer. Logic based systems aim to use semantics in order to handle and answer directly complex queries that couldn't be handled by classic search engines [19].

1.1 Motivation

Achieving such results implies a good representation and a good analysis of natural language. However, current approaches such as Natural Language Processing (NLP) or First Order Logic (FOL) based Theorem Provers do not provide - at least, for now - acceptable solutions. In fact, NLP focuses more on robustness and computation speed than on accurateness. Even if some new algorithms can be used for complex inferences on natural language such as neural network in deep learning, they often work as black boxes. A result is given, but no information proving it are available. On the other side, theorem provers can deal with complex proofs, but the problem of designing a reliable and robust automated translation process from natural language to FOL remains.

1.2 Goal

The objective of this Master Thesis is to enable complex querying as described with logic based systems. More precisely, it aims to discuss a system able to query relations between complex concepts in bio-scientific text based on graph representations.

As current solutions are not suited for such querying, Natural Logics has been chosen to represent natural language and to enable queries over it.

1.3 Approach

In order to support and implement our researches, the logic programming language Prolog has been chosen. Prolog programs are composed of a set of rules expressed as *if conditions* which operates over constants, variables and predicates which are processed using tree search. By querying one of the rules, Prolog returns all possible solutions satisfying it. This makes Prolog particularly suited for theorem proving, expert systems and to deal with computational linguistic - which was the main interest of its lead creator Alain Colmerauer.

The first part of the project focuses on describing the essence of Natural Logics based on ontologies and traditional syllogistic logic in order to understand how information is carried by natural language (II). The second part of the project focuses on defining a Core Natural Logics (III). Even if its scope remains too limited for a concrete application, Core Natural Logic enables to discuss important concepts, classical problems related to language processing and important design decisions. Based on these choices, a set of extensions for Natural Logics to cope with the most common structures of scientific texts will be defined (IV). However, these extensions implies an increased complexity of the Natural Logics which brings along some ambiguity problems. The fifth part of the project proposes a solution based on ontological analysis of concepts (V). Defining an algorithm in order to generate graphs from sentences is the focus of the sixth part (VI), before focusing on defining an efficient algorithm for path finding working over Natural Logics' graph representation (VII). Finally, the implementation of the approach defined will be confronted with concrete examples from bio-scientific texts (VIII) in order to analyze and assess its strength and weaknesses (IX).

CHAPTER 2

Knowledge in Natural Language

This section introduces the concepts of *ontology* and *syllogistic logic* in order to represent the knowledge carried by natural language. A first approach for Natural Logics arises from the will of working over this knowledge based on syntactic structure representations for ontology structured knowledge bases.

2.1 Ontology and concept representations

The term ontology refers to the philosophical study of the nature of being, becoming, existence or reality. Ontology's application to computer science can be defined as a formal explicit description of concepts in a specific domain of discourse [32]. Particularly, its importance has been recognized in various research fields as knowledge engineering, knowledge representation, language engineering or database design. As explained by [21], the creation, analysis and use of ontology especially refer to adopting a highly interdisciplinary approach based on philosophy and linguistics. The use of high level of generality and of a clear and specific vocabulary is essential.

"Ontology are designed to capture what is universal across a given domain while databases primarily captures what is particular."

Barry Smith, International Conference on Biomedical Ontology in Lisbon, 2015.

Some basic classical applications of ontology are the definition of partonomy (also named meronomy) and taxonomy to classify ontological concepts and define relations among them. Taxonomy focuses on defining a hierarchy of discrete groups of biological organisms based on shared characteristics, defining then a hierarchy. Partonomy, on the other side, focuses on part-whole relationships. Defining ontological concepts, relations between them and concrete instances of these concepts results in the creation of ontology-structured knowledge bases.

In order to perform semantic search, the interest of this thesis especially relies on the use of generative ontology, meaning on the use of an ontology able to represent hierarchical structures and part-whole relations, but that is also able to accommodate with arbitrarily complex concepts. Generative ontologies result in a hierarchy[1] of

[1]Note that ontology aren't necessarily strictly hierarchical in the way that some concepts might appear simultaneously at different positions in the ontology.

concepts whose properties can be modified, restricted or expanded based on binary relations working over them [3]. They offer then an increased modularity and expressivity.

For this project, the domain of discourse considered is related to bio-medical sciences and more specifically to the human insulin production. The interest relies then on a definition of a domain specific ontology focusing on concepts related to human metabolism and hormonal production.

2.2 Traditional syllogistic logic

To embrace natural language's knowledge, generative ontologies should be able to cope with concept relations defined by natural language and especially with verbs (for instance, *produce, regulate, create...*). A first step towards this objective can be achieved by the ability to reflect traditional syllogistic logic.

Especially, generative ontologies should be able to reflect the underlying structures of Aristotelian natural logic syllogistic sentence forms which can be derived from the square of opposition[2] (figure 2.1).

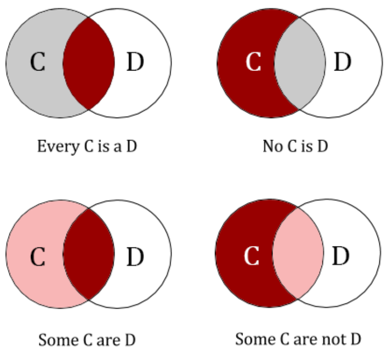


Figure 2.1: Square of opposition.

While most formal ontologies basically apply the class inclusion relation *isa* in their hierarchy and thus can deal with the structure *every C is a D*, they cannot directly deal with the other ones. The negation of concepts is often represented by introducing appropriate classes based on class disjointness [31].

[2]Respectively named: *universalis affirmativa* expressing concept inclusion (top left corner), *universalis negativa* expressing concept disjointness (top right corner), *particularis affirmativa* expressing overlapping concepts (bottom left corner) and *particularis negativa* expressing that the concept C cannot only be reduced as a subconcept of D (bottom right corner).

Letting negation aside, the structures defined by the square of ontology are especially useful since they can be used to derive a finite set of 4 structures underlying natural language sentences. The following presents them with examples:

- ($\forall\exists$) default sentence structure: *[every] betacell produce [some] insulin*
- ($\forall\forall$) specific case for some definitions: *[every] treatment inhibits [every] infection*
- ($\exists\exists$) specific case for passivization[3]: *[some] insulin is produced by [some] betacell*
- ($\exists\forall$) specific case for properties: *[some] treatment inhibits [every] infection*

Due to the trouble of ontologies to represent copular sentences from the square of opposition, most ontologies also fail in dealing with these structures in a simple way, even if representing them appears as a requirement for logic frameworks aiming at reasoning over natural language.

2.3 Description Logics for ontology representation

This section aims to give a quick overlook at Description Logics, which represents a commonly used approach to discuss ontologies, especially in web semantics, in order to illustrate the limitations presented previously.

2.3.1 Introduction to Description Logics

Description Logics - Description Logics refers to a family of decidable fragments of FOL. They can be described as modelling languages containing a formal semantic for unambiguous language representation, but also enabling the deduction of new information from facts explicitly stated. This process named *inference* is the basis of reasoning and is the main difference between DL and other modeling languages such as UML [26]. Many types of DL with varying complexity and decidability exist and are used depending on the intended application.

Building blocs - Description Logics is based on a clear distinction between individuals, concepts (also referred to as classes) and roles. Classes are defined as formulae or predicates with only one free variable [20].

Boxes - Three different types of knowledge are described: ABox, TBox, RBox [26]. The ABox captrures knowledge between individuals and concepts such as concept assertions: *Mother(julia)* or individual and roles such as role assertions: *parentOf(julia,john)*. The TBox describes relationships between concepts such as concept subsumption *Mother* $\sqsubseteq$ *Parent* or equivalence *Human* $\equiv$ *Person*. The RBox refers to properties of roles such as role inclusion *ParentOf* $\sqsubseteq$ *AncestorOf*.

Representation of concepts - Description Logics is based on an extended use of recursions and modifications of classes. This results in a broad definition of classes that

[3]Active and passive sentence forms are discussed in detail when considering Extended Natural Logics.

can be either compound or simple (table 2.1).

Notation	Informal interpretation
C ⊓ D	class at the intersection of C and D
C ⊔ D	class as union of C and D
¬ C	class being the negation of C
∃ R.D	class being something that R D
∀ R.D	class being all the things that R D

Table 2.1: Examples of classes in DL.

A specific interest is given to compound concepts since they can be used to represent relative clasues. Formally, a compound concept is composed of the intersection of a concept C with a binary algebraic operator $\exists\ R.D$ based on a binary relation R and a second concept D. For instance, *cell* is an atomic concept while *cell-that-produce-insulin* is a compound term where $C =$ cell, $R =$ produce, $D =$ insulin and can represented as: cell ∩∃produce-insulin. The composed concept $C \cap\exists\ R.D$ is considered as a subclass of C which results from the application of the restrictive binary relation $\exists\ R.D.$

Description Logics also consider the existence of a top ontology $\top$, to which all classes and instances are related, and of a bottom ontology $\perp$.

2.3.2 Limitations of Description Logics

Description Logics enables then to deal with partonomic relations, class inclusions, but also recursive structures which makes it particularly suited for working with with ontologies. However, it still shows some limitations when trying to reproduce natural language.

Syntax - DL arises from the will of representing ontologies in a more readable way than FOL, but its syntax remains really different from natural language. The problem of designing an automatic translation process from and to natural language remains then.

Natural language structures - Moreover, DL does not seem to be suited to handle the four syntactic structures defined previously that originating from Aristotelian Logics and that are underlying natural language sentences. If, Description Logics deals straightforwardly with the case ∀∃, using inclusion and existential quantification, the processing of other cases appears as really tricky if not impossible.

Representation of propositions - Finally, the choice of leaving DL is also related to its awkward representation of natural language. In fact, in DL, propositions are only represented through concept inclusion and ascription of properties to concepts and do not consider relations between concepts [31]. For instance[4], *betacell that produce insulin reside in pancreas* would be represented as $cell \cap \exists produce.insulin \sqsubseteq \exists residein.pancreas$.

[4]The case of *universal restriction* could also be mentioned here. In fact, the construction

The obtained DL representation is then closer to *cell that produce insulin [are something that] reside in pancreas* than the original sentence.

These reasons motivate the will of developing an alternative approach for semantic analysis on natural language.

2.4 Natural Logics as a generative ontology representation of natural language

Natural Logics aims to overcome the limitations of DL by focusing on a representation closer to natural language where the emphasis is moved from the concepts onto the relation terms. The idea is to define Natural Logics as based on fragments of natural language where the reasoning can be directly conducted without any further representation.

2.4.1 Emphasizing relation terms

A decisive element in order to work over natural language is to consider a new representation of sentence structures from syllogistic logics where the emphasis relies on the relation terms. This implies to abandon the strict corpula form of traditional syllogistic logic, in favor of logical sentences accepting a main verb. The idea of this approach is then to allow the definition of binary relation between classes when working over natural language instead of only considering a disguised method to represent them based on class inclusions and on the ascription of properties to concepts [31].

2.4.2 Syntactic structures for ontology structured knowledge bases

In order to represent the structures underlying natural language and to put the emphasis on relation terms, the syntactic form $S = Q1\ C\ R\ Q2\ D$ has been defined for Natural Logics fragments. The terms C and D represent concepts that can be either atomic or compound. Concepts definition based on natural language semantics is one of the main topic of this project in the next chapters. R represents a relation term that is composed of a transitive verb eventually suffixed by some preposition. $Q1$ and $Q2$ are quantifiers that can be either existential like *some* or universal like *every*. Thus, based on different combinations of these quantifiers, this syntactic form enables to represent the four different types of structures originating from syllogistic logic.

$\forall produce.insulin$ represents all individuals able to produce insulin, but also individuals that are not producing anything, which might not be the intended meaning in natural language for "all people that have daughters". This difference is motivated by the underlying FOL representation $\forall x(produce(x, y) \rightarrow insulin(y))$ and the definition of the implication relation. The actual meaning of $\forall produce.insulin$ is then *everything do not produce something different than insulin.* To represent elements that only produce insulin, and thus produce at least something, the syntax is $(\exists produce.\top) \cap (\forall produce.insulin)$ [26]

Expressing natural language relations - Among them, a particular attention is given to *every C R some D* which is considered as default case and which can be represented in predicate logic as follows: $\forall$x, C(x) $\rightarrow$ $\exists$y, (R(x,y) $\wedge$ D(y)). This structure enables to cover the underlying default structure of natural language sentences. It can be used to cover sentences such as *betacell produce insulin.*

Ontological relations - This syntax can also be used in order to cover typic ontological relations. By introducing a keyword for concept inclusion such as *is* or *isa*, this syntactic structure enables to deal with hierarchical structure rules. In fact, the *universalis affirmativa* proposition can be formulated as *every C is some D*, defining the class inclusion *C isa D*. This notation enables as well to deal with partonomic relations by using the binary relations such has *has-part* and *is-part-of.* For instance, representing that beta-cell are a part of the pancreas, can be done using: *every pancreas has-part some beta-cell* or *every beta-cell is-part-of some pancreas.* It is then possible using syntactic structures to deal with partonomic relations using both active and passive voice.

Conclusion - Using this structure, it is then possible to define the basis of a model for a generative ontology able to bring together basic entity relationships models (dealing with partonomic and class inclusion relations), binary relation inspired from verbs, but also considering compound concepts formed by composition of lexicalized classes and relationships [3]. These concepts and relations can then been included inside a generative ontology so that they can be used to serve semantic purposes.

CHAPTER 3 Core Natural Logics

After this discussion of the essence of Natural Logics based on generative ontologies focusing on relation terms and aiming at carrying natural language's knowledge, this part focuses on defining Core Natural Logics. Core Natural Logics is a first approach to Natural Logics whose scope is limited and that will be extended further on in order to deal with larger parts of natural language. This part mainly aims at explaining basic, but crucial, notions related to Natural Logics and at introducing and discussing some major design decisions.

3.1 Scope of Core Natural Logics

Scientific texts - Natural Logics focuses on representing scientific texts, and does not aim at discussing other types of texts. This relies on the assumption that the ontological analysis of scientific texts is less ambiguous since they strive for clarity and then do not make an extensive use of stylistic figures of speech. For instance, Natural Logics does not aim at dealing with metonymies and metaphors.

Dropping quantifiers - Natural Logics is based on the syntactic representation of the *universalis affirmativa* proposition. In fact, the structure *every C R some D* is the one underlying most natural language sentences. For instance, *betacells produce insulin* can be represented in First-Order-Logic as: $\forall$x, Betacell(x), $\exists$y (R(x,y) $\wedge$ Insulin(y)) and thus, matches this pattern. In order to simplify the notation and to be closer to natural language, it has been decided to drop, in the default case, the quantifiers in the syntactic notation of Natural Logics. As a consequence, the structure *every C R some D* is transformed into *C R D* and a natural language sentence like *betacells produce insulin* is then translated as *betacell produce insulin*. Later on, when dealing with structures possessing other quantifiers, they will be explicitly represented.

Handling sentences independently - Another important decision concerning Natural Logics concerns the processing of texts where each sentence or proposition is considered independently from all others. This decision is motivated by the consideration that knowledge is contained in sentences themselves. The relations between sentences are believed to only reflect the argumentative strategy of the author, which might make the understanding for the reader easier, but does not contain more knowledge than in a set of independent sentences.

Scope ambiguity - When dealing with predicate logic, a classical problem encountered is scope ambiguity and especially quantifier scope ambiguity. This phenomenon arises in sentences containing more than one quantifying noun phrases, as explained in [8] with the example: *every boxer loves a woman.* In fact, this sentence can be translated into First Order Logics in two ways:

1. ∀x, (Boxer(x) → ∃y (Woman(y) ∧ Love(x,y)))
2. ∃y, (Woman(y) → ∀x (Boxer(x) ∧ Love(x,y)))

The sentence has then two possible interpretations. On the one hand, each boxer loves a woman and these women are potentially different (1). On the other hand, it can be considered that each boxer loves a woman and this woman is unique for all boxers (2). However, it can be seen that the first case actually includes the second one and represents then a weak equivalent. Thus, Natural Logics solves quantifier scope ambiguity by considering that the first interpretation which always holds.

Negation - Natural Logics does not consider an explicit use of negation. Negation arises from the closed world assumption. If a concept or a relation between concepts is not represented in the knowledge base, then this concept or this relation does not hold[1]. Thus, a concept like $\neg C$ or a relation like in the sentence $C \neg\ R\ D$ won't be represented. However, the subconcepts C and D are added to the knowledge base since they can contain further knowledge especially in case of compound concepts.

Concept representation - Core Natural Logics focuses on representation of simple concepts. This includes atomic concepts like *cell*, only considering this type of concept leads to Atomic Natural Logics, but also compound concepts of type $C \exists\cap R.D$ where C and D are atomic concepts. Natural Logic concepts relies on existential import, meaning that each concept represented possesses at least one physical instantiating. Empty concepts are then not considered. The limitation to simple concepts enables to discuss concept representation further on without complexifying the problem too much. However, this assumption won't be considered when dealing with extended versions of Core Natural Logics.

3.2 Representation of Natural Logics propositions

3.2.1 Representations as Definite Clause Grammar

Based on Prolog's Definite Clause Grammar (DCG) notation (Appendix C.1) a context free grammar for the syntactic structure of Core Natural Logics propositions $C\ R\ D$ can be defined as follows:

[1]It is known that closed world assumption (CWA) can lead to contradictions when not dealing with definite clauses. For instance from $A \wedge B$, using CWA, it can separately be deduced $\neg A$ and $\neg B$, and then that $A \wedge B$ does not hold. However, this won't be an issue for Natural Logics since only definite Horn clauses will be used [35]

```
% Syntactic form.
s −−> np, r, np.

% Nominal phrase.
np −−> n.
np −−> n, [that], r, np.

% Noun − concept.
n −−> [N], {lex(N, concept)}.

% Relation.
r −−> [R], {lex(R, relation)}.
```

In this representation,*np* stands for Nominal Phrase and can represent either atomic or compound concepts [1]. The relation term *r* is used to represent binary relations between concept terms. These relations can be either a transitive verb like *produce* or *reside_in* or the class inclusion keyword *isa*.

This set of DCG clauses represents then a grammar for Core Natural Logics that can be used to verify if sentences are syntactically correct.

```
?− s([cell, that, produce, insulin, reside_in, pancreas],[]).
true.

?− s([cell, that, produce, produce, reside_in, pancreas], []).
False.
```

3.2.2 Definition of a parser for Core Natural Logics

However, when true, this predicate s/3 does not explicitly give any information on why it holds. This can be fixed by transforming this set of clauses into a top down parser by adding a new argument representing the structure to each predicate.

```
% Syntactic form.
s(s(N1,R,N2)) −−> np(N1), r(R), np(N2).

% Nominal phrase.
np(np(N))−−> n(N).
np(np(N1,R,N2)) −−> n(N1), [that], r(R), np(N2).

% Noun − concept.
n(n(N)) −−> [N], {lex(N, concept)}.

% Relation.
r(r(R)) −−> [R], {lex(R, relation)}.
```

By proceeding as follows, a left first top down parser, usually called *LL-Parser*, can be defined. The result of a request is now as follow:

```
?− s(X, [cell,that,produce,insulin,residein,pancreas],[]).
X = s(np(n(cell), r(produce), np(n(insulin))), r(residein),
     np(n(pancreas)))
```

3.3 Concept graph representation

Based on its grammar, Core Natural Logics aims at transforming sentences into oriented graphs called *concept graphs.* In these graphs, nodes represent concepts that can either be simple or compound while edges can represent class inclusion or binary relations. This graph representation aims at bringing out explicitly the information carried by natural language in a way that allows the application of inference rules and graph search [3], [2], [31], [1], [4]. This representation will also be used in this chapter to discuss important design decisions.

3.3.1 Representation of simple propositions

The representation of a simple proposition, *Cterm R Cterm'*, is composed of two nodes related by an edge. For instance, *betacell produce insulin* and *betacell isa cell* are represented as follows.

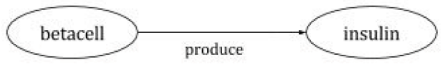


Figure 3.1: Concept graph representation of transitive verbs.

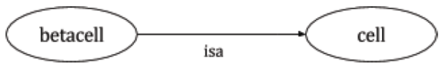


Figure 3.2: Concept graph representation of class inclusions.

3.3.2 Concept graph representation of compound terms

This representation can also be used to represent compound terms involving relatives clauses by decomposing them in basic propositions. For instance, *cell that produce insulin* is decomposed in *cell-that-produce-insulin isa cell* and *cell-that-produce-insulin produce insulin.* These propositions are then transformed individually into graphs and, based on the unicity of concepts in the graphs, identical concepts are merged together. The sentence is thus, represented as follows.

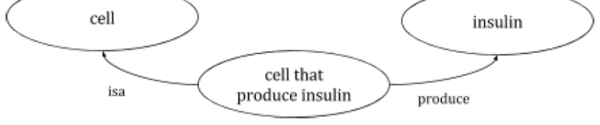


Figure 3.3: Representation of complex propositions.

The definition of an automated process decomposing Natural Logics propositions into concept graphs is discussed in chapter 6.

3.4 Natural language rules

This part focuses on defining formal inference rules reflecting human inferences (how humans draw conclusions) and working over natural language.

3.4.1 Monotonicity and logical inferences rules

Natural Logics focuses on inferences involving monotonicity based on class inclusions, defined through the keyword *isa* which is reflective (X *isa* X) and transitive (if a X *isa* Y and Y *isa* Z, then X *isa* Z). Based on the notation *isa*, it is then possible to define the relations of transitivity, inheritance and generalization as described pictorially in the figure 3.4 [2]:

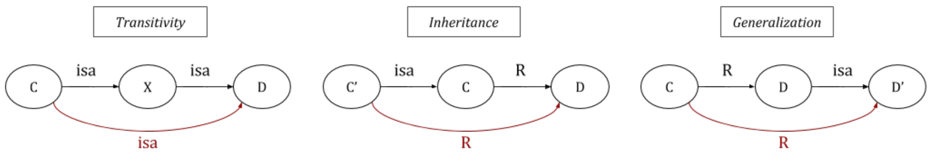


Figure 3.4: Graphical representation of inference rules.

These relations are straightforward to deal with for the human brain. They are all based on the concept of monotonicity [27]. Natural Logics aims to explain inferences by involving monotonicity in which the concepts (nouns, verbs..) or constraints (relations, adverbs, quantifiers...) expressed are expanded or contracted. For instance, the sentence *every beta-cell produce some insulin* some elements can be expanded while preserving truth, but not contracted, and oppositely for others. On the one hand, the terms *insulin* can be expanded to *hormone*. Its polarity is then positive. On the other hand, the terms *every* or *beta-cell* can be respectively contracted to *some* and *healthy beta-cell* while preserving truth. Their polarity is then negative. Polarity is discussed as based on a relation of entailment and on the definition of monotonicity of semantic elements.

Entailment - Entailment relation is defined as follows [27]:

- if c and d are of type t (truth values, for instance: variables), c $\sqsubseteq d \Leftrightarrow c \rightarrow$ d
- if c and d are of type e (entities, for instance: constants), c $\sqsubseteq d \Leftrightarrow$ c = d
- if c and d are of type $<\alpha,\beta>$ (functional, such as predicates), c $\sqsubseteq d \Leftrightarrow \forall a \in \alpha, c(a) \sqsubseteq d(a)$
- if c $\not\sqsubseteq$ d and d $\not\sqsubseteq$ c, then c # d

Monotonicity - Most linguistic expressions are upward-monotone such as *residein* since *residein pancreas* $\sqsubseteq$ *residein organ* is obtained based on *pancreas* $\sqsubseteq$ *organ*. However some important constructions, such as negation, restrictive quantifiers (no, few, at most n), restrictive verbs (lack, fail, prohibit) and certain adverbs are downward monotone, for instance: *not residein organ* $\sqsubseteq$ *not residein pancreas*. A few other expressions are non monotone such as superlative adjectives and quantifiers.

3.4.2 Inference rules elicitation

In Core Natural Logics, subjects of proposition can be expanded while objects of proposition can be expanded [1], [2], [4]. The following clauses[2] express how new facts can be generated based on *isa* relations.

```
fact(T,Csub,R,D) :-isa(Csub,C), fact(T,C,R,D).
fact(T,C,R,Dsup) :-fact(T,C,R,D), isa(D,Dsup).
fact(T,Csub,R,Dsup) :-isa(Csub,C), fact(T,C,R,D), isa(D,Dsup).
```

New facts are then obtained by applying monotonicity rules on the concepts C and D from a fact *fact(T,C,R,D)* contained in the knowledge base where T represents the type of the facts. In fact, types in Core Natural Logics are distinguished in *observations* and *definitions*.

As an example, let's consider the following knowledge base:

- fact(observation, beta-cell, produce, insulin).
- subclass(healthy-beta-cell, beta-cell).
- subclass(insulin, hormone).

Applying the monotonicity based inference rules would then generate the new facts:

- fact(observation, healthy-beta-cell, produce, insulin).
- fact(observation, beta-cell, produce, hormone).
- fact(observation, healthy-beta-cell, produce, hormone).

3.4.3 Subsumption rules

This decomposition of sentences in graphs calls for the use of a new logical inference rule, named *subsumption rule* [2]. This rule aims at calculating logically relevant missing class inclusion from compound terms. This can imply, if needed, the creation of new concepts nodes.

Upward monotonicity - In Core Natural Logics, subsumption is based on upward monotonicity on the modifier and on the concept modified.

Upward monotonicity on modifier - From the compound concept *C that R D* and the inclusion relation *D isa D'*, the concept *C that R D'*, and the relations *isa(C that R D, C that R D'), isa(C that R D', C), fact(definition, C that R D', R, D')* are created

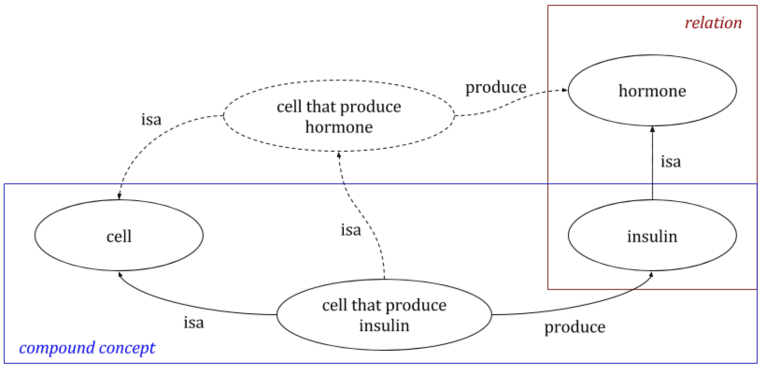


Figure 3.5: Example of upward monotonicity on concept modifier.

if missing in the concept graph. An example is shown figure 3.5:

Upward monotonicity on concept - From the compound concept *C that R D* and the inclusion relation *C isa C'*, the concept *C' that R D* and the relations *isa(C that R D, C' that R D), isa(C' that R D, C'), fact(definition, C' that R D, R, D)* are created if missing. An example is shown in the figure 3.6.

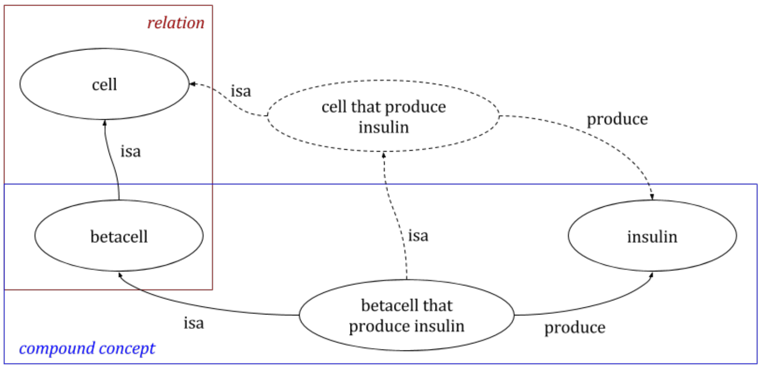


Figure 3.6: Example of upward monotonicity on concept modified.

Upward monotonicity on both - Finally, upward monotonicity can in some cases be applied on both modifier and referent. This results in the more complex case that can

[2]Only the last rule could have been needed if *isa* had been defined as reflexive in the implementation. This isn't the case in order to avoid potential infinite loop due to the transitivity and reflixivity of *isa*

be seen in figure 3.7.

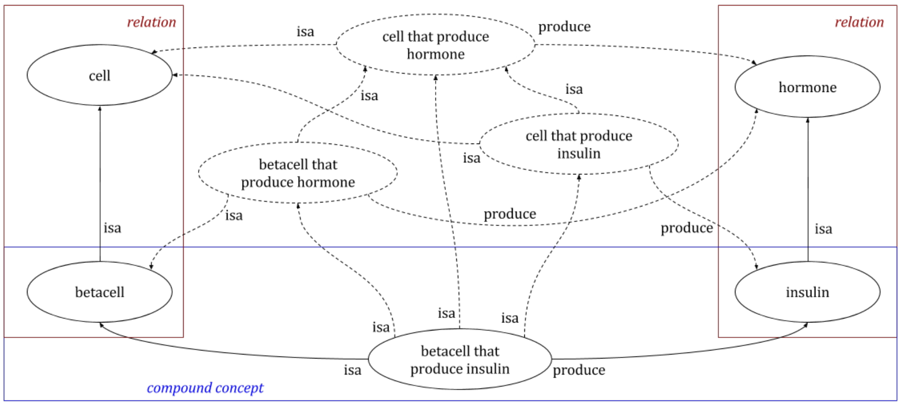


Figure 3.7: Example of upward monotonicity on concept modified and concept modifier.

Downward monotonicity - However, downward monotonicity is not considered due to the existential import assumption. For instance, from the compound concept: *C that R D'* and the inclusion relation: *D isa D'*, the concept *C that R D* is not created, since it might not exist and then wouldn't fit with the existential import assumption made for Natural Logics.

This computation allows to speed up search processes by extending graphs and to create more reliable short paths. Later on, this approach will be extended to other concept modifiers.

3.5 Discussing left recursive parsing issue

3.5.1 Left recursive issue for compound term parsing

In order to get a fully recursive syntactic structure able to deal with all complexity of compound terms, the grammar should be modified as follows:

```
 % Fully recursive noun phrase.
np(np(N)) --> n(N).
np(np(N1,R,N2)) --> np(N1), [that], r(R), np(N2).
```

However, this would result in a left-recursive rule that creates an infinite loop in most language including Prolog. In fact, the second rule, while firing the clause np(N1), will call itself indefinitely.

A classical solution in order to avoid defining such loops is the use non terminal predicates. Non terminal predicates are predicates that are not designed to be rendered to the user. Their purpose is purely related to processing.

```
% Right recursive noun phrase.
np(np(N)) --> simple_np(N).
np(np(N1,R,N2)) --> simple_np(N1), [that], r(R), np(N2).
simple_np(N) --> n(N).
```

However, this extra predicate adds one more layer in the definition of the syntactic structure which does not reflect any linguistic element or concept. This somehow encounters the initial goal of using DCG to have a simple representation. Worst, this predicate does not help the grammar to distinguish some sentences since left-recursion is not handled anymore.

The complex noun phrase *cell that produce insulin that reside_in pancreas* can be used as example. Without considering any semantic analysis for unambiguous parsing, this complex noun should remain syntactically ambiguous and should then be interpreted in two ways by the grammar of Core Natural Logics:

1. cell that produce (insulin that reside_in pancreas)
2. (cell that produce insulin) that reside_in pancreas

The first sentence is based on the application of a right-recursive rule whereas the second one is generated by a left-recursive one. However, when considering the previous Prolog implementation, the parsing result omits, as expected, the second possibility:

```
?- np(X, [cell, that, produce, insulin, that, reside_in, pancreas], []).
X = np(n(cell), r(produce), np(n(insulin), r(reside_in), np(n(pancreas)))) ;
false.
```

3.5.2 Alignment as solution for left recursive noun phrases

This parts discusses an approach to handle left recursion in Natural Logics parse trees from a purely computational point of view. Considering, the graph search objective of Natural Logics, this section especially aims at showing how depth limited recursion can be avoided in favor of a solution based on concept alignment[3].

Depth limited recursion - Depth limited recursion is an intuitive solution to prevent infinite recursion. However, this solution isn't the best if not applied properly. On the one hand, if the depth limit is not big enough, some solutions won't be found. On the other hand, if the depth limit is too high, it could introduce a slow down of the performances of the parer, especially in the case of Natural Logics since this approach aims to deal with a larger set of features for natural language. To be considered as a solution, a dynamic computation of the depth limit should be introduced. Solutions

[3]From a purely semantic point of view, the use of alignment for left recursion can still be discussed. However, it provides here a feasible solution considering the graph-search objective of Natural Logics discussed in this thesis.

based on depth restrictions with respect to input length and adapting to the current position in the tree parser have been developed to obtain a polynomial answer time[4] [18].

Weak equivalence - Weak equivalence, which has already been shortly discussed when dealing with scope ambiguity, could also be seen as a potential solution for left recursive structures. Formally, weak equivalence could be defined as follows: *A sentence S1 is weakly equivalent to S2 if S1 entails S2, which means that S1 defines a case that is more general than the sentence S2.* When using S1 instead of S2, truth is preserved, even if some information is lost.

Alignment - A weak equivalent representation for left recursive structures can be defined using aligned structures. In fact, instead of using a left recursive NP such as *(C that R D) R' D'*, it is possible to consider having both relative clauses referring to the concept *C*. The use of alignment is weakly equivalent to the left recursive structure since it entails it: if *C* realizes *R' D'*, then, based on the previously defined inference rules, *(C that R D)*, which is a subconcept of *C*, does it as well. However, this approach seems to drop on the way the restriction that only *C that R D* is able to satisfy the relation *R'* with the concept *D'*.

Disambiguation - Expressing alignment can be done through the use of explicit alignment keywords like commas or like *and*[5]. This has the advantage of explicitly defining the scope of relative clauses. As a consequence, it is now possible to have two distinct notations to represent aligned structures and nested structures. They should be understood as follows:

1. Cell that produce insulin that residein pancreas $\rightarrow$ insulin residein pancreas
2. Cell that produce insulin and that residein pancreas
   $\rightarrow$ cell residein pancreas $\wedge$ (cell that produce insulin) residein pancreas

Limit - Alignment provides then a method to represent without ambiguity two distinct structures for compound concepts. Especially, alignment provides a weak equivalent to represent left recursive structures when using DCG clauses, but this approach seems to drop some information on the way. This information should particularly be discussed in the context of concept graph representation.

3.5.3 Equivalence discussion based on concept graphs

Recalling that the goal of Natural Logics is to work over concept graph representations, this section discusses a potential equivalence between aligned and left recursive structures based on concept graph representation. The following representations show two different types of lines. The ones in plain line represent direct decomposition. The

[4]Proposition of an algorithm for ambiguous left recursive grammar based on a set of parser combinators

[5]Alignment without any explicit keyword based on ontological analysis will also be discussed in chapter 4 and 5.

other ones represent concepts and relations created using inference and subsumption rules.

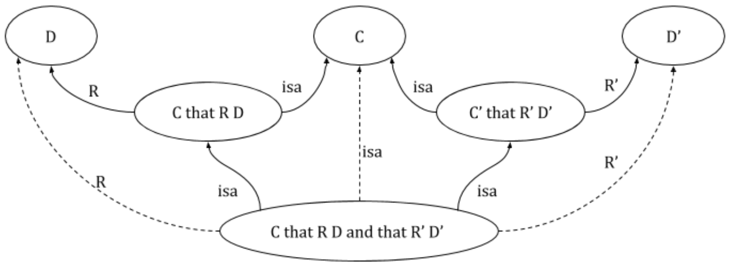


Figure 3.8: Concept graph representation of an Aligned NP.

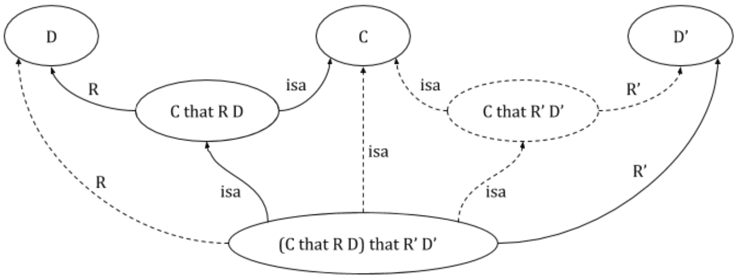


Figure 3.9: Concept graph representation of a Left Recursive NP.

As a direct consequence of weak equivalence, it can be seen that the concept graph representation of the aligned structure (figure 3.8) entails than the representation of the left recursive one (figure 3.9). In fact, the aligned structure contains all concept of the left recursive structure, but contains also one more concept *C' that R' D'* and the following three relations:

1. isa((C that R D) that R' D'), C that R' D')
2. isa(C that R' D', C)
3. fact(definition, C' that R' D', R', D')

However, after applying inference rules and subsumption rules on aligned and left recursive structure, both structure actually have the same concept graph representation. Subsumption enables then to create the missing concept *C that R' D'* based on the concept *(C that R D) that R' D'* and the relation *isa(C that R D, C)*. The missing relations can then be created straightforwardly based on this new concept.

From a concept graph point of view, when considering subsumption, aligned structures can be considered as a solution for Natural Logics in order to represent left recursive structures.

3.6 Definition of Natural Logics

From this introduction to Core Natural Logics, a general definition of Natural Logics can be defined as follows:

> “Natural logics consists of stylized fragments of natural language where reasoning can be conducted directly by natural reasoning rules reflecting intuitive reasoning in natural language.” [1]

In a more detailed way, Natural Logics aims to extract and make use of the knowledge carried by natural language by defining a generative ontology which especially reflects extended Aristotelian syllogistic logic with verbs. Based on ontology structured knowledge bases and simple logical rules applied to natural language, Natural Logics creates proofs by incrementally editing expressions of natural language by using rules specifying under which conditions semantic expansions or contractions preserve truth. This way, Natural Logics ambitions to obtain a robustness similar to theorem provers while using a language representation whose syntax is closer to natural language. Natural Logics is then able to reflect natural language underlying structure and its translation from and to natural language is easier to process.

CHAPTER 4
Extending Core Natural Logics

After this presentation of Core Natural Logics, the objective of this section is to extend the previously defined grammar in order to capture larger fragments of natural language. Especially, Natural Logics grammar should be able to deal with most natural language structures represented in scientific texts to perform semantic analysis.

The choice of these structures has been motivated by their occurrence in scientific articles and their semantic importance. Most of the examples used in this chapter have been made up in order to match the needs of the discussions. Some of them were also taken from the reference text on insulin [39]. In this case, the provenance of the sentence is explicitly stated.

4.1 Extensions applied on concepts

The first type of extensions for extended Natural Logics applies on noun phrases and the concept they represent.

4.1.1 Prepositional Phrases

Similarly to relative clauses (RCs), prepositional phrases (PPs) can be defined in Natural Logics. They involve prepositions in order to modify a noun with a restrictive value. Their structure is then as follows *C PREP D* where *C* and *D* represents concepts while *PREP* represents a preposition (in, of, by, ...) such as in *cell in pancreas*. In this case, the notation *PP-in* would be used. From this construction, two relations can be extracted. One class inclusion: *(cell in pancreas) isa cell* and another one involving the preposition and bringing together two concepts: *(cell in pancreas) in pancreas*. On this point prepositional phrases are similar to transitive verbs.

Prepositional phrases are valuable in the way that their meaning can be considered as explicit[1], in opposition to other structures such as compound nouns or possessives.

[1] In order to return results as sentences, returning the preposition as a link between concepts can be enough. However, to explicit the underlying semantic relations carried by the preposition, some more work might be needed. Section 5 gives more information about it in its disambiguation approach.

4.1.2 Compound nouns

Definition - Compound nouns refers to noun phrases that are made up of two or more words like *blood glucose* or *liver production.* Compound nouns are particularly interesting for linguists since it seems that their meaning cannot be reduced to the sum of the meaning of its components taken individually [10].

4.1.2.1 Syntax of compound nouns

Types of compound nouns - Three types of compound nouns can be distinguished: close compound nouns when words nouns are merged together (*toothpaste*), hyphen compound nouns when an hyphen is used (*well-being*) and space compound nouns when a space is used (*blood glucose*). While, hyphen and close compound nouns aren't an issue for Natural Logics, space compound nouns cannot be handled as such and they should be integrated in Core Natural Logics' grammar.

4.1.2.2 Semantic of compound nouns

Semantic challenges - Compound nouns are semantically challenging structures. On the one hand, they combine words without any grammatical marking so that their relation is implicit. On the other hand, the semantic behaviour of words implied in compound noun constructions do not remains invariant. For instance with the word *safe* takes two opposite meaning when dealing with *child safe*, meaning *safe for children* and *shark safe*, meaning *safe against sharks.* It seems then than not simple and straightforward rule can be expressed regarding the composition and the meaning of compound nouns, while dealing with them still remains intuitive and straightforward for the human brain. Their creation and understanding seems to be the result of a cognitive process whose activity overcomes the meaning of both elements and put a links between them based on the representation of them [10].

Rule elicitation - Even if they might not completely cover the semantic around compound nouns, some rules can be defined:

- XY compounds are asymmetric; they do not contribute equally and X appears to be a predicate of Y, defining X Y as a subset of Y. For instance, a *boathouse* is an house and a *house-boat* is a boat.
- the relation between X and Y can be relative to the general schema displayed around Y or the way is integration of X in Y is cognitively process.

Schema driven approach - Some claim that there exist a schematic algorithm underlying the meaning of compounds. This approach is based on the creation of schemas (figure 4.1) around the conception of the words involved in compound noun construction [10].

For instance, based on the definition and the cognitive representation that everyone has of a station, this word can be combined to create compound nouns based on different aspects of it: its equipment, its purpose or its location. The denomination

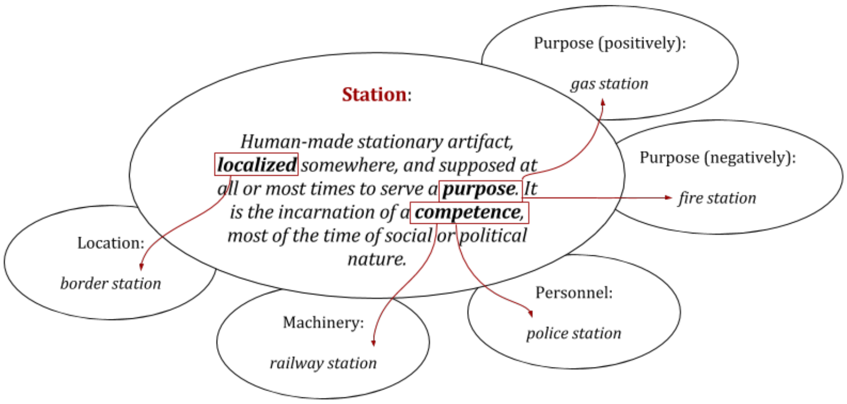

Station:
Human-made stationary artifact, localized somewhere, and supposed at all or most times to serve a purpose. It is the incarnation of a competence, most of the time of social or political nature.
Purpose (positively):
gas station
Purpose (negatively):
fire station
Location:
border station
Machinery:
railway station
Personnel:
police station

Figure 4.1: Schema around *station* in its related compound nouns.

chosen is motivated by what appears most salient about the nature of the station in question. As a consequence, a fire station refers to the iconical representation of a burning house, while a police station refers to the uniform of policemen which is more salient than the abstract notion of crime [2].

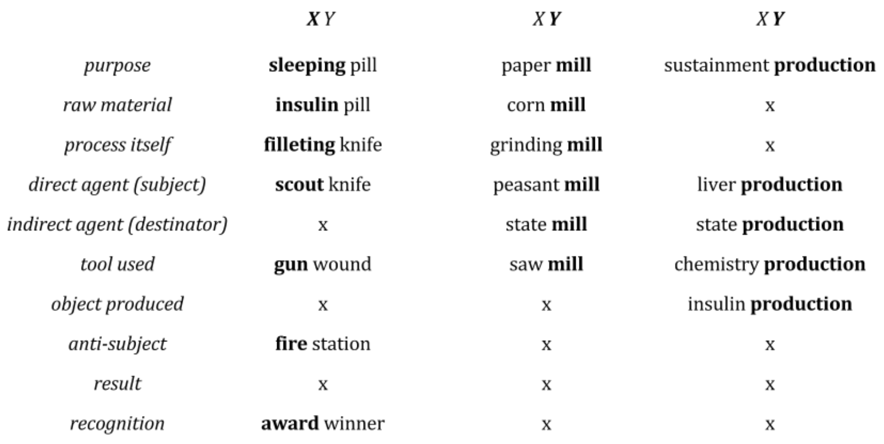
Purpose oriented events - Its application onto purpose oriented events is interesting and is particularly suited to work on compound constructions involving processes such as *production*. However, this approach isn't limited to this case. Compound constructions (XY) can be made around a non-process element Y, X refers then to a part of Y's function, production, use or purpose. In this situation, X and Y can both fulfill one of the following functions: purposes, raw materials, agents, instruments, etc and combine freely together as seen on the figure 4.2.
Even if the number of potential combination might seem confusing, this approach is powerful in the way that it covers a large number of compound nouns. Especially, this approach is particularly interesting in the domain of discourse of this thesis since it contains a lot of processes.

This approach is similar with the ontological discussions around concepts that are presented in chapter 5, but remains harder since an extra step is added for non-process objects by using a metonymy to refer to their purpose, production, function or use.

4.1.3 Possessives

Definition - In English, possessive structures are concept in the form of *C's D* where *C* and *D* are concepts, like in *Mary's book*.

[2]Other similar representation of a police station can be done like *handcuff station* or *anti-crime station* and would be understood by most speakers.



	X *Y*	*X* ***Y***	*X* ***Y***
purpose	**sleeping** pill	paper **mill**	sustainment **production**
raw material	**insulin** pill	corn **mill**	x
process itself	**filleting** knife	grinding **mill**	x
direct agent (subject)	**scout** knife	peasant **mill**	liver **production**
indirect agent (destinator)	x	state **mill**	state **production**
tool used	**gun** wound	saw **mill**	chemistry **production**
object produced	x	x	insulin **production**
anti-subject	**fire** station	x	x
result	x	x	x
recognition	**award** winner	x	x

Figure 4.2: Purpose oriented events for compound nouns.

Implicit information - Possessive structures usually have a restrictive function and describe a class inclusion: *(C's D) isa D*. However, they also contain another information on the concept *C* which relation can express a relation of:

- **Possession:** for instance *Mary's book*,
- **Part-whole relation:** for instance *Mary's leg*,
- *Affiliation:* for instance *Mary's arrival*,
- *Creation:* for instance *Mary's music*.

Information elicitation - Possessive relations contain then an implicit relation information whose nature is found depending on the type of element mentioned. Ontological analysis of the concepts *C* and *D* could be used here as well. Especially, this could be done if *D* is a process by following the same purpose oriented approach as for compound nouns. As an example, *liver's production* would then obviously be recognized as a creation relation where *liver* is the agent.

Remaining ambiguity - However, this wouldn't be sufficient to cover all cases. For instance, the term *Mary's music* could refer to the music that Mary has in her collection of CDs and MP3 files (possession), but could also refer to the music she has composed (creation). The first case would be transformed into a PP as the *music of Mary*, while the latter would be transformed into the *music by Mary*. Background knowledge about the context (*is Mary a musician?*) is then needed in order to disambiguate the situation. No obvious relation can be drawn between creation and possession since it is possible to possess something that has been created by someone else and the other way around. A solution in this case, can be to consider a weak equivalent since both creation and possession relation imply an affiliation relation that can be used.

4.1.4 Adjectives

Definition - Adjectives refer to words that modify nouns or pronouns by ascribing them some additional properties or qualities. They act as concept modifiers able to combine recursively with nouns [34].

A classic approach for adjectives is to consider them as purely restrictive and letting the basic properties of the noun unmodified. Their application on a concept would then result in the creation of a subconcept of the original noun. For instance, an *anabolic hormone* is a subconcept of *hormone* which has the property to be *anabolic*. However, this isn't always true, since some adjective as *fake* do not follow this pattern: a *a fake Picasso painting* isn't a *Picasso painting*. As a consequence, different types of adjectives can be defined based on the way the modify the noun they compose with (figure 4.3) [29] [24] [34].

- **intersective** (or extensional): they represent the most common class of adjectives. The denotation of the composition of an intersective adjective and a noun, is the intersection of the denotation of its constituents. Examples: blue, french, anabolic.
- **subsective** (or intensional): they represent a common class of adjectives where the denotation of the composition of the adjective and the noun defines a subclass of the noun. Examples: big, small, cold. They are related to a non objective appreciation of the situation. They can be degree adjectives (tall, short, big) or evaluative adjectives (good, skillful, remarkable). For degree, meaning comes from context. For evaluative, comes from the noun.
- **non-subsective**: they represent a class of adjectives for which the denotation of the composition of the adjective and the noun might or not intersect with the denotations of the noun. Examples: potential, possible, unlikely. They are linked to a form of uncertainty. For instance, a possible minister can or cannot be a minister.
- **privative**: they represent a class of adjectives for which the denotation of the composition of the adjective and the noun does not intersect with the denotation of the noun. Examples: virtual, fake, former. It includes then the example mentioned previously with the *fake Picasso painting*.

Predicativity - Another distinction between adjectives depending on if they are syntactically predicative or not. An adjective is syntactically predicative when it appears alone as the complement of a copular verb such as *be*.

Intersective, subsective and privative adjectives can be predicative while non-subsective are not, but no fixed rule be extracted out of it. For instance, *criminal* and *environmental* which are intersective are not predicative, while *anabolic* or *red* are.

1. (inter. - pred.) an anabolic hormone $\rightarrow$ the hormone is anabolic
2. (inter. - non-pred.) an environmental lawyer $\rightarrow$ *the lawyer is environmental
3. (sub. - predicative) a tall cat $\rightarrow$ the cat is tall

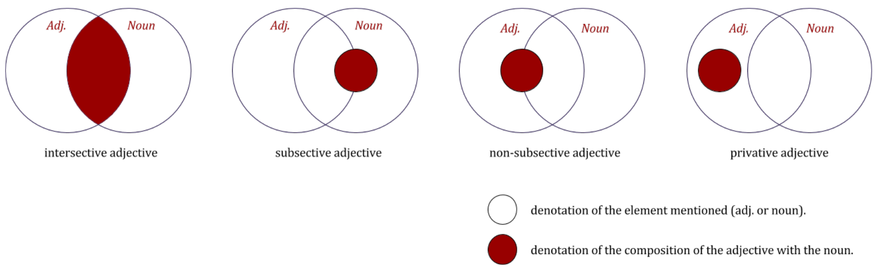

Adj.
Noun
intersective adjective
subsective adjective
non-subsective adjective
privative adjective
denotation of the element mentioned (adj. or noun).
denotation of the composition of the adjective with the noun.

Figure 4.3: Graphical representation of the different types of adjectives.

4. (sub. - non-pred.) my old school → *the school is old
5. (priv. - predicative) a fake painting → the painting is fake
6. (non-sub. - non-pred.) a former president → *the president is former

Adjectives in Natural Logics - In order to properly deal with adjectives from a semantic and syntactic point of view, two types of information have to be considered: the semantic behaviour (intersective, subsective...) and the syntactic predicativity. [3]. Each adjective should then be presented like in this example: *lex(tall, adjective, subsective, predicative)*. Superlatives and comparisons which are based on adjectives are not handled for now.

4.1.5 Appositions and parenthetical clauses

Paraphrases are commonly encountered in natural language and more specifically in bio-scientific texts. Their purpose is typically to explain or clarify the text or the word placed before the paraphrase. Paraphrase are usually introduced with a declaratory expression expressing the transition to the paraphrase. This transition can be signaled using structures such as *who is*, parenthetical relative clauses, a phrase inserted within another phrase, or appositions [40].

In opposition to all other extensions seen until now, they do not have a restrictive value.

Two types of paraphrases can be distinguished depending on the text they paraphrase. On the one hand, paraphrases on words can be distinguished, for instance *insulin, a hormone, regulate body* which adds an additional information. On the other hand, paraphrase can be applied on proposition or sentences. For instance, *the light was red,*

[3]Some argues however for an equivalence of syntactic predicativity and semantic predicativity. An adjective is semantically predicative if and only if: It is of type <e, t> (modulo contextually-specified information). It combines with any nominal it modifies prenominally via Predicate Modification. It is then possible to only use semantic equivalence to define it. Further categories of adjectives needs then to be discussed as nominal adjectives (criminal, electrical), modal adjectives (former, alleged), event-manner (hard worker, beautiful) or predicate-evaluating (mere, common). The reference [11] gives more details about it.

that is, the train was not allowed to proceed. However, this case won't be handled since the approach used in Core Natural Logics is to handle each proposition independently from all others and then, from the context of the text. However it can be assumed that this case could be solved by some specific pre-processing of the text.

Linguistically it is important to recall two differences in order to avoid misinterpretations [1]. The use of *that* which includes a restriction with the use of *which* and *who* whose purpose is to generalize. Furthermore, the difference between relative clauses preceded by a comma, as discussed previously, and those which are not should be handled carefully. A relative clause preceded by a comma adds additional information to the concept. For instance, *gland, that*[4] *synthesizes substances* informs that by definition all glands synthesizes insulin and is considered equivalent to *gland which synthesizes substances.* On the contrary, a relative clause that is not preceded by a comma modifies the concept by restricting it. For instance, *gland that synthesizes substances* appears as a subclass of gland implicitly meaning that all glands are not synthesizing substances.

From [1] [4], Natural Logics consider appositions and parenthetical clauses as follows:

- Apposition bounded by commas: *Cterm , [a/an] Ctermappo, R Cterm" ,*
- Parenthetical relative clause: *Cterm , [which/who] Rtermappo Ctermappo, R Cterm" ,*

Both of these cases would be transformed into a conjunction of two propositions. On the one hand: *Cterm isa Ctermappo* and *Cterm R Cterm".* On the other hand: *Cterm Rtermappo Ctermappo* and *Cterm R Cterm".*

For now only these types of appositions are considered. For instance, PPs are not considered since they can be confused with PP phrases and it is expected that they would be turned into a parenthetical RC. For instance *betacell, in pancreas, produce insulin* can be turned into *betacell, which reside in pancreas, produce insulin.*

4.1.6 Fully recursive structure for compound concepts

Finally, the last extension on concepts (noun phrases) for Extended Core Natural Logics is to allow fully recursive structure for compounds concepts using alignment as a solution for left recursion and without limitations on the number of concept modifiers (RC, PP, adjectives, parenthetical clauses, appositions and possessives) involved.

4.1.6.1 Syntactic order

This implies to define rules to avoid syntactically incorrect structures and, as much as possible, potential ambiguities. First, it is possible to distinguish pre-modifiers from post modifiers. Pre-modifiers are modifiers placed before the noun. In English, this

[4]From a pure grammatical point of view, the use of *which* would be expected, but this structure remains currently used though.

covers adjectives, possessives and noun composition. On the other hand, post-modifiers are placed after the noun and cover RC, PP and appositions.
From an analysis of the order in which they are usually placed, the following generic structure for modifiers can be modeled[5]:

$$\{\text{possessive}\}\ \{\text{adj}\}^*\ \{\text{compound}\}^*\ \text{noun}\ \{\text{apposition}\}\ \{\text{RC} \mid \text{PP}\}^*$$

For instance, this structure is underlying noun phrase example *liver's automated and continuous glycogen production, which is essential, in the metabolism, that regulate the body and by glycogenesis* detailed in the figure 4.4.

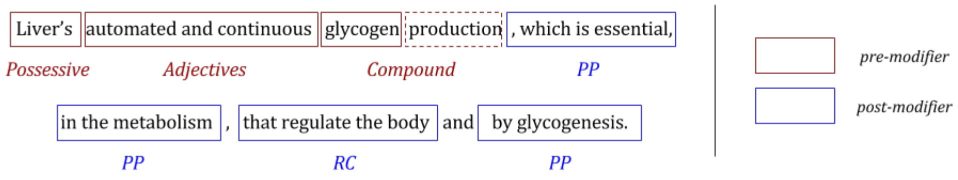


Figure 4.4: Syntactic structure of noun phrases.

This structure covers most of the noun phrases from natural language. Some exceptions are still not covered such as adjectives modified by nouns like in *virus infected cell.*

The modifiers referring to the noun are all aligned. No specific keywords are needed in order to express the alignment of pre-modifiers, while post-modifiers can be associated with commas and the keyword *and* to express the alignment. As a consequence, the following structures are always considered aligned:

- C post-modifier-1 and post-modifier-2
- C post-modifier-1, post-modifier-2 and post-modifier-3
- C post-modifier-1 and post-modifier-2 and post-modifier-3

However, the absence of these keywords does not always imply having a nested structure. In fact, in natural language, some structure can be aligned. For instance, the sentence *production in pancreas by glycogenesis* is an aligned structure since glycogenesis cannot refer to pancreas and refers then to production. The syntax should then consider that, even without explicit alignment, the structure might semantically be aligned.

4.1.6.2 Syntactic composition

After discussing the ordering of the modifiers, the focus is now to define how modifiers can compose together syntactically. The aim is to discuss how concepts involved in

[5]Syntax explanation: The curly brackets, {adj}, indicates an optional element, here an optional adjective. The symbol * indicates that this element can be placed many times. Finally, the symbol | is similar to a logical or.

modifiers can be themselves modified by other modifiers.

Adjectives - Adjectives are not involving any other concept than the one they modify. No restriction are here necessary.

PP, RC & Appositions - The concepts involved in these structures can be freely modified. For instance, *synthesis in (liver's glucose production that regulate the metabolism)* shows a sentence where the concept involved in the RC is modified by a possessive, a noun composition and another RC.

Compound Nouns - The noun C in a compound C D cannot be modified. For instance, when applying an adjective on C, it would be understood as referring to D. *huge glucose production* is actually more likely to be understood as a *huge production of glucose* and not as a *production of huge glucose.*

Possessives - The noun C in a possessive C's D can only be modified by noun composition and by adjectives. For instance, in the phrase *huge glucose production's behaviour*, the adjective and the composition refers to production, while in the phrase *production's huge behaviour*, the adjective refers to the behaviour. Especially, C cannot accept post modifiers since they might be confusing. For example, *(glucose that regulate body)'s production* cannot be expressed, because it would be understood as *glucose that regulate (body's production).* It is important to highlight that the use of a compound noun with a possessive should be understood as a left recursion. For instance *glucose production's behaviour* refers to *(glucose production)'s behaviour* and not to *glucose (production's behaviour).*

4.2 Extensions applied on relations

Since Natural Logics gives a decisive role to expressing relations, extensions can be defined for them as well.

4.2.1 Prepositional verbs

This allows Natural Logics to deal with verbs containing a preposition like located in using two distinct verbs and not a specifically created concept *locatedin* or *located_in.* On a large knowledge base, it also avoids creating one specifically dedicated relation for each possible association of a verb with a preposition (*located_under, located_in, located_above* etc). Moreover this also help Natural Logics to look more like free natural language formulations.

4.2.2 Passive forms and passivization

Definition - A sentence is said to be passive when the subject undergoes the action. For instance the sentence *insulin is produced by betacell* is passive since its subject, *insulin*, is acted on by the verb. Passive forms are syntactically recognizable by the use of the BE and the past participle of the verb showing the action followed, optionally by a PP-by.

Equivalence - In Natural Logics, active and passive voices are not considered to be purely equivalent. For instance, for the sentence *betacell produce insulin*, meaning *[every] betacell produce [some] insulin*, the corresponding passive voice in natural language is *insulin is produced by betacell.* However, its underlying quantification is *[every] insulin is produced by [some] betacell*, which is not logically equivalent. From this sentence, the information that all betacell produce insulin is missing. The meaning of the original active sentence, is actually closer to *[some] insulin is produced by [every] betacell* which is considered as a weak equivalent. As a consequence, the distinction between passive voices should be kept and the representation of passive voices should be integrated among the features of Extended Core Natural Logics.

Passivization - The process of turning an active sentence into passive voice is called passivization. The passive sentence is constructed in three steps [5]. The transitive verb of the active sentence is turned into its passive form using the verb *be* and the past participle form of the verb in the active sentence. The direct object of the active sentence is then used as subject for the passive sentence, while the subject of the active sentence is transformed into an adverbial PP-by which is placed at the end of the sentence. During the passivization, it is not mandatory to mention the agent. When performing the passivization, the underlying quantifiers of the sentence are transformed into ∃∃ and not ∃∀. In fact, the structure ∃∀ cannot be inferred from the active one.

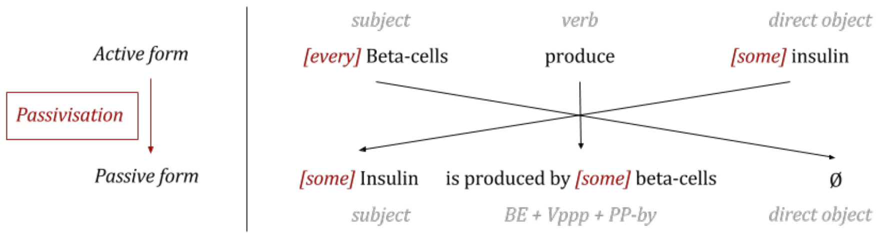


Figure 4.5: Passivization process.

Activation - The activation process is made on the other way around and has similar properties as the passivization discussed above.

From this, two types of sentences can then be distinguished:

- sentences that where present as such in the original text and whose structure is then ∀∃.

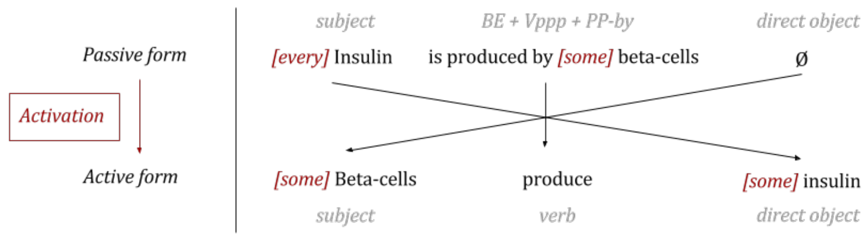


Figure 4.6: Activation process.

- sentences that result from passivization or from activation and whose structure is then ∃∃.

The use of these two different structures (∀∃ and ∃∃) for passive sentences makes here an echo to the motivations of abandoning Description Logics which cannot handle ∃∃ structures in favor of Natural Logics[6].

4.2.3 Nominalization

Definition - Nominalization refers to the process of producing a noun from another part of speech. For Natural Logics, nominalization of transitive verbs representing relations is discussed.

$$Rterm \iff Concept$$

Voice equivalence - As Natural Logics does not consider passive and active sentence as being equivalent, nominalization should then be able to reflect it. Active sentence can be nominalized using possessive forms while passive sentences are nominalized using a PP-by. The table 4.1 present examples of nominalization for the active sentence *Liver produce glucose* and its passive voice *Glucose is produced by the liver.*

Representation	Active Form	Passive Form
Verbal form	*Liver produces glucose*	*Glucose is produced by the liver*
Gerund / PP-by	*Liver's production of glucose*	*Production of glucose by the liver*
Compound noun	*Liver's glucose production*	*Glucose production by the liver*
Compound nouns	*Liver glucose production*	

Table 4.1: Example of Nominalization.

Even if it wouldn't put emphasis on any active or passive voice, a nominalization containing two compounds nouns such as *liver glucose production* could also exist.

[6]The passivization and activation process are processed during the graph search when rendering the results, see chapter 7.

Equivalency of representations - As seen previously, nominalization involves potentially a combination of compound nouns, possessives and PP-by. Based on the discussion of these extensions, some nominalization should be favored since the use of PP contains explicit information while compound nouns and possessive structures doesn't. If a representation encountered had to be chosen, *Production of glucose by the liver* for passive voice would be the best one while *Liver's production of glucose* would be the best active one. However this distinction won't be considered for now in favor of the use of two PPs.

4.2.4 Adverbs and adverbial Prepositional Phrases

Definition - The term *adverb* refers to a word that describes or gives more information about a verb, an adjective, another adverb or a phrase. Adverbs appear then to be related to a broad number of lexicon entries and to a multitude of semantic behaviours. This section is essentially based on [28] and its dedicated chapter on adverbs.

Types of adverbs - From this variety, different classes of adverbs can be defined. A distinction is made between predicational adverbials considered as gradable predicates (like amazingly, similarly or cleverly), in opposition to adverbs quantifying over participants or events (like possibly, necessarily or always). Among predicational adverbs, a distinction is then made in a few main classes[7]:

- **Event adverbials**: they modify the event described in its manner or in its result.
- **Subject oriented**: they give information on the subject performing the action.
- **Speaker oriented**: they give information about the act of speech reporting the action.
- **Frame setting**: they describe the context in which the action happen.

Syntax & semantic - The classification of an adverb in one of these classes depends on the adverb itself, but also on its position when used in the proposition. In the sentence: *Happily, Floyd would happily play the tuba happily*, the adverb *happily* shows different meaning and its instances belongs to many classes. Especially, the work of McConell-Ginet (82) shows that an adverb placed before the verb is generally subject oriented (*Louisa rudely answered Patricia*), while it acts as a manner adverb when placed after the verb (*Louisa answered rudely Patricia*).

Similarity with adjectives - From a semantic point of view, the first attempts to deal with adverbs were trying to relate their semantic behaviour to the one of adjectives. This approach arises from the intuition that the syntactic similarity between adverbs and adjectives[8] implies also some similar semantic behaviour. This is also motivated

[7]Jackendoff (1972) and Maienborn (2001)

[8]Syntactically, the construction of a significant part of adverbs is linked to adjectives since it is possible to turn an adjective into an adverb by adding the suffix *-ly*, like with *quiet* $\rightarrow$ *quietly*

by the behaviour of adverbs which seems to act as predicate modifiers[9], but also based on the entailment patterns they can generate[10] that is similar to adjectives.

However, this approach has some limitations. First, it doesn't explain why some adverbs show different behaviours when drawing inferences. In fact, subject-oriented adverbs seem to be sensible to the phenomenon of referential opacity while other adverbs like manner adverbs aren't. The phenomenon of referential opacity appears when the differences between the beliefs of the subject in his representation of the world and the world as it is, have consequences on the inferences that can be drawn. For instance, the following inference is invalid[11]:

1. Oedipus intentionally married Jocasta.
2. Jocasta is Oedipus's mother.
3. therefore: Oedipus intentionally married his mother. (invalid)

On the other hand, in the case of manner adverbs such as quietly, this opaque context does not appear:

1. Oedipus quietly married Jocasta.
2. Jocasta is Oedipus's mother.
3. therefore: Oedipus quietly married his mother. (valid)

Secondly, some adverbs cannot match the semantic behaviour of adjectives. From the previous examples, the adverb intentionally seems then to quantify over world interpretations compatible with Oedipus attentions and can then be considered somehow as subsective. From this, the adverb *quietly* is then not subsective. Since most adjectives are subsective or intersective, quietly is then expected to be intersective. If it was, the proposition *Oedipus quietly married his mother*, would then be translated as $quietly(x) \wedge married(x, y) \wedge motherOf(x, y)$ where x refers to Oedipus. However, quietly actually refers to the marriage and not to Oedipus. It seems then that manner adverbs like *quietly* cannot be interpreted semantically in a similar way as adjectives.

Davidson approach - The current approach bringing back together subject oriented and manner adverbs relies on Davidson's view of adverbs (1967) as modifiers of the properties of the event described by the proposition[12] and on the definition of a higher

[9]Predicate modifiers return the same type that the element they apply on. Defining the type of elements adverbs actually apply on (proposition or VP) has been and is the subject of debates.

[10]For instance, from the sentence, *Floyd ran awkwardly quickly*, it can be inferred that *Floyd ran awkwardly*, that *Floyd ran quickly* and that *Floyd ran.*

[11]This phenomenon is also used in order to argue that adverbs are VP-modifiers and not acting on the proposition. In fact, the inference of Oedipus intentionally marrying is mother is not valid and shows then that referential opacity apply on the object of the sentence. However, this opaque context does not happen when dealing with subject of the sentence. From the sentences: *Oedipus intentionally married Jocasta* and *Oedipus is the son of Laius*, the inference *The son of Laius intentionally married Jocasta* is valid.

[12]This is motivated by the fact that it is semantically possible to refer to the event (giving him a name), to refer to the event (using a pronoun like it) and also to ascribe properties to events.

abstract verb *act* that defines the event whose properties are modified by the adverbs. A sentence like *Louisa rudely departed* can then be turned into *Louisa acted rudely in departing* which can be expressed as follows:

$$\lambda e.\exists e'[cause(e')(e) \wedge agent(e) = Louisa \wedge depart(Louisa)(e')] \wedge rude(manner(e)).$$

The rude departure is then considered as an event that was rude and that was caused by the departure of Louisa who was the agent. With this approach, it is then possible to bring together manner and subject oriented adverbs (and potentially, others like result adverbs) in a way showing a behaviour similar to adjectives.

4.2.4.1 Adverbs for Natural Logics

Scope definition - Natural Logics focuses on adverbs used as modifiers of relations. Thus, speaker oriented adverbs or quantification adverbs won't be considered. The focus is then set on locative and manner adverbs.

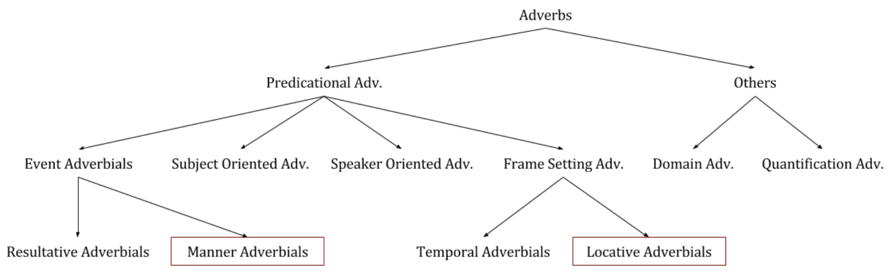


Figure 4.7: Scope of Adverbials in Natural Logics.

Subject oriented adverbs are not considered for now in order to work with only one syntactic position for adverbs: after the verb. This position is also used for frame setting adverbs, even if they usually appear at the beginning or at the end of the proposition, in order to avoid potential ambiguities.

Concept Graph Representation - Since Natural Logics aims at representing propositions as concept graphs, it is particularly important to discuss adverbs representation. The representation of one adverb or adverbial PP could be done by branching the adverbial on the relation modified (figure 4.8).

However this representation cannot be extended easily to relations modified by many adverbs or adverbial PPs. As a consequence, four different designs for concept graph representations have been proposed (figure 4.9). Some were fitting with concept graphs principles, but were lacking in generality by not showing all the potential inferred knowledge (figure A). Others were introducing either a new type of nodes for modifiers (figure C) or relations of relations (figure B and D). None of these approaches were then completely satisfying.

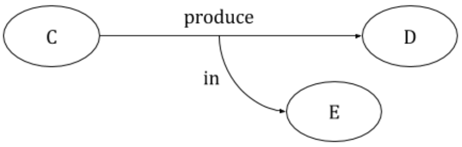


Figure 4.8: Example of concept graph of a proposition containing one adverbial PP.

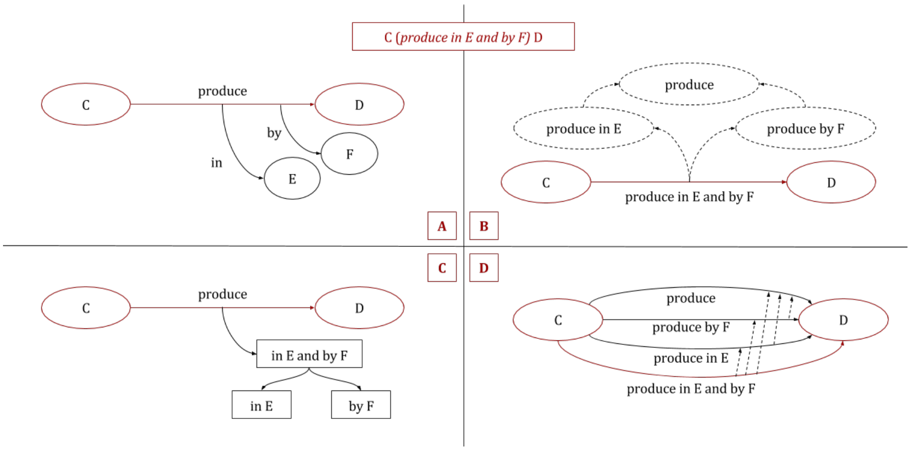


Figure 4.9: Potential concept graph representations for propositions with multiple adverbial PPs.

Finally, the solution chosen for adverb representation is to rely on a the underlying Davidsonian event express using a nominalization of the proposition. Each proposition involving adverbials is then *attached* to its nominalized form. In this nominalization, adverbial PP remains as PP while other adverbs are transformed into their corresponding adjective[13]. The figure 4.10 shows one example where the sentence *liver produce via glycogenesis glycogen* is turned into the concept *production of glycogen via glycognesis by liver* which is then decomposed as a concept graph.

4.2.5 Expressing conditions

An approach to deal with conditions is to transform them as adverbial PPs using nominalization. For instance, the sentence from the insulin article [39] *when glucose concentrations in the blood are high, beta cells secrete insulin into the blood* can be transformed as *beta cells secrete insulin at high glucose concentration in the blood.* This way, the meaning can be captured by re-using already defined data structures. However, in order to keep track of this, a new property should then be added to relations to distinguish general cases (without conditions) from specific cases (with

[13]For instance, by deleting the suffix *-ly*: *highly produced* $\iff$ *high production*

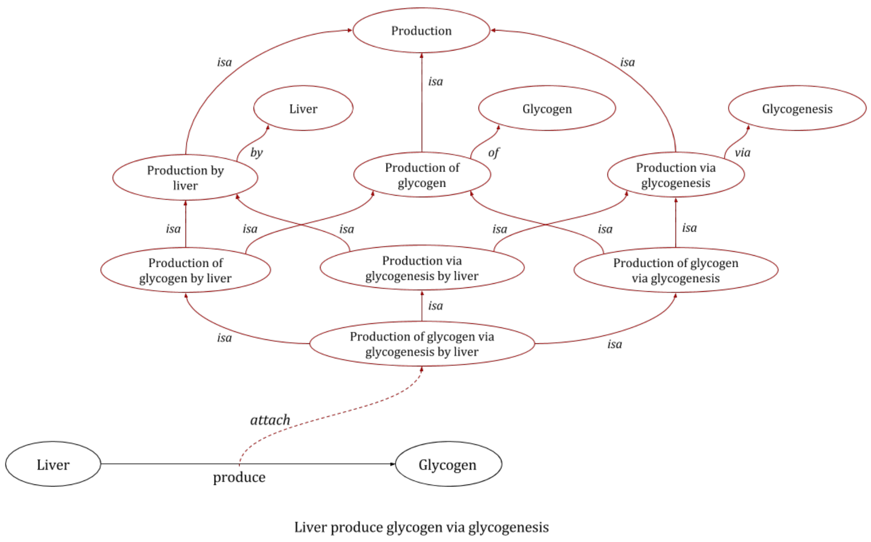

Production
isa
isa
isa
Liver
Glycogen
Glycogenesis
by
of
via
Production by liver
Production of glycogen
Production via glycogenesis
isa
isa
isa
isa
isa
isa
Production of glycogen by liver
Production via glycogenesis by liver
Production of glycogen via glycogenesis
isa
isa
isa
Production of glycogen via glycogenesis by liver
attach
Liver
produce
Glycogen
Liver produce glycogen via glycogenesis

Figure 4.10: Concept graph for multiple adverbial PP using a nominalization.

conditions). This is especially important to consider this distinction during the graph search.

4.3 Extensions applied on propositions

This section discusses extensions that are neither applying exclusively on concepts or on relations. Based on an introduction to plural formations, this section defines rules to handle conjunction and disjunction of concepts as subjects and objects of relations. Conjunctions and disjunctions applied inside RCs or PPs are not discussed here, since they can be handled by distributing them into two RCs or PPs respectively.

4.3.1 Plural formation

As stated by Franconi [15], a distinction concerning plurals can be made between classes and collections. Classes are involved in sentences such as *betacells are cells* and are then transformed in FOL using predicates: $\forall x, betacell(x) \rightarrow cell(x)$. Collections represent aggregations of objects, called members, and are represented using instances and not using predicates. As a consequence, in order to express the sentence: *the Beatles are led by John Lennon*, Franconi first defines the members of the Beatles as part of a collection: $\ni (beatles, john), \ni (beatles, ringo), \ni (beatles, paul), \ni (beatles, george)$ and then he uses a predicate on the collection: $led\_by(beatles, john)$. It appears then that classes can be handled by Natural Logics without any new extension, but collections cannot.

Including collections as plurals in Natural Logics involves that predicates may have plural values as well as singular values. This is called plural quantification [41] and is achieved by adding properties issued from set theory into the models and by multigrade predicates or relations (meaning elements whose arity is not fixed).

Theory of collections - Franconi [15] argues that dealing with plurals isn't pure set theory, but refers actually to collection theory. In fact, each collection has specific properties referring to the collection as a whole. Thus, two collections, even composed of the same members, are not considered as equal. As a consequence, the extensionality principle[14], one of the bases of set theory, does not hold.

Readings - From this statement, each plural sentence involving a collection can have multiple readings [12]. For instance, the sentence *the men lifted the piano* is usually understood as three men lifting the same piano, while the sentence *the men played the piano* is understood as men playing separately on different pianos. However, the opposite interpretation could be true as well. Only the context can define which of the reading was correct. Three types of reading can actually be described.

- Distributive reading: attributes a property to each member of the collection.
- Collaborative reading: attributes a property to the collection as a whole.
- Cumulative reading: each element of the collection satisfies individually or belongs to a collection that satisfies the relation. The collaborative and distributive readings appear as subcases of the cumulative reading.

As an application of this, Covington [12] demonstrates that a sentence such as *the three farmers fed the three donkeys* has at least nine potential readings:

1. The farmers as a group fed the whole group of donkeys (1 act of feeding, collaborative reading for farmers, collaborative reading for the donkeys).
2. Each of the farmers fed the whole group of donkeys (3 acts, distributive, collaborative).
3. Two farmers together and one farmer separately fed the whole group of donkeys (2 acts, cumulative, collaborative).
4. The farmers as a group fed each of the three donkeys individually (3 acts, collaborative, distributive).
5. Each of the three farmers fed each of the three donkeys (9 acts, distributive, distributive)
6. Two farmers together and one farmer separately fed each of the three donkeys (6, cumulative, distributive).
7. The farmers as a group fed two of the donkeys together and one separately (2, collaborative, cumulative).

[14]The extensionality principle or axiom of extension can be expressed as follows [38]: 'Given any set A and any set B, if for every set X, X is a member of A if and only if X is a member of B, then A is equal to B'. This is translated into FOL as $\forall A \, \forall B \, (\forall X \, (X \in A \iff X \in B) \Rightarrow A = B)$.

8. Each of the three farmers fed two of the donkeys together and one separately (6, distributive, cumulative).
9. Two farmers fed two donkeys together and one donkey separately, then the third farmer fed two donkeys and one donkey separately (4 acts, cumulative, cumulative).

Respective reading - All these cases are special cases of the last one. Based on this example, Covington argues, that one reading is missing and cannot defined based on combinations of other three: when each farmer is feeding exactly one donkey. He names this case respective reading. While other readings relate collections to collections, the respective reading focuses on individuals defining 1-to-1 relations between objects. This reading is usually chosen in sentences such as *the mothers give the books to the children.*

Conclusion - Plural formation can in general be reduced to two readings: the cumulative reading and the respective reading when many plural formations appear in the sentence. However, these readings can still induce the creation of a lot of ambiguities, especially when collections contain a larger number of members.

4.3.2 Conjunctions

Natural Logics aims to tackle plural formations by focusing at first on collections of two elements. The collection is created as a conjunction of two concepts. In this case, Natural Logics always consider a distributive reading. The collective one is considered as going beyond the scope of Natural Logics approach[1],[4].

This choice is logically motivated by the underlying predicate logic structure. In fact, the sentence *Cterm1 and Cterm2 R Cterm"* is quantified as $\forall x(Cterm1(x) \wedge Cterm2(x) \rightarrow \exists y(R(x,y) \wedge Cterm"(y)))$, which is equivalent to $\forall x Cterm1(x) \rightarrow \exists y(R(x,y) \wedge Cterm"(y)) \wedge \forall x Cterm2(x) \rightarrow \exists y(R(x,y) \wedge Cterm"(y))$.

Similarly, the sentence *Cterm R Cterm"1 and Cterm"2* is quantified as $\forall x(Cterm(x) \rightarrow \exists y(R(x,y) \wedge (Cterm"1(y) \vee Cterm"2(y))))$, which is equivalent to $\forall x Cterm(x) \rightarrow \exists y(R(x,y) \wedge Cterm"1(y)) \wedge \forall x Cterm(x) \rightarrow \exists y(R(x,y) \wedge Cterm"2(y))$.

As a consequence, *Cterm and Cterm' produce insulin* means that both Cterm and Cterm' can individually produce insulin. This proposition can then be transformed into a conjunction of propositions as *Cterm1 R Cterm"* and *Cterm2 R Cterm".* Same applies on conjunctions used on the object [4].

In case of multiple plurals on both the subjects and the objects, the respective reading won't be chosen in favor of the distributive reading. The subjects and objects are distributed twice. As a consequence, a proposition like *Cterm and Cterm1 R Cterm2 and Cterm3* is decomposed into four propositions, *Cterm R Cterm2*, *Cterm R Cterm3*, *Cterm1 R Cterm2* and *Cterm1 R Cterm3*, covering at the same time the two ones that would result from a respective reading.

4.3.3 Disjunctions

Extended Natural Logics also aims at dealing with disjunctions of two concepts [4]. The approach is different from conjunctions since disjunctions might imply that some concept don't satisfy the relation described by the proposition. For instance, *Cterm1 or Cterm2 R Cterm3* may refer to a case where *Cterm2 R Cterm3* does not hold. Since they can introduce errors, disjunctions applied on the subject are then considered irrelevant. The use of disjunctions is then restricted to objects as in *Cterm R Cterm1" or Cterm2"*.

Disjunctions on objects are also not handled similarly to conjunctions of objects. A sentence containing a disjunction such as *Cterm R Cterm1" or Cterm2"* cannot be decomposed as a conjunction or a disjunction of two proposition as discussed with conjunctions. In fact, it might insert a proposition that does not hold in the knowledge base. The chosen approach is then to use a common supremum, if available in the knowledge base. More precisely, this common supremum is defined as *Cterm1" isa Ctermsup* and *Cterm2" isa Ctermsup*. For instance, a common supremum for *alpha cell* and *beta cell* can be *cell*. A proposition like *insulin is produced by alpha cell or beta cell* would then be transformed into *insulin is produced by cell*.

A good handling of disjunctions also includes to find the optimal order in which sentences should be processed. This aims at avoiding the scenario where the common supremum needed for a disjunction is not in the knowledge bases when the sentence is processed, but only appears later on. This scenario would cause the failure of the parsing of the sentence while it should have actually succeeded.

4.4 Natural Logic Grammar for extended Core Natural Logics

The aim of this section is to turn the extensions discussed previously into a grammar for Natural Logics.

4.4.1 Natural Logic Grammar

Based on the previous set of extensions, the syntactic representation of Natural Logics sentences can be represented using the following natural logic grammar.

```
Prop ::= Csubject R Cobject
Csubject ::= Cterm
Csubject ::= Cterm {and} Cterm
Cobject ::= Cterm
Cobject ::= Cterm {and | or} Cterm
Cterm ::= {Poss} {ADJ}∗ {NOUN | CompNoun} {App | ParClause} {RelClause | PrepPhrase}∗
RelClause ::= Rpronoun R Cobject.
PrepPhrase ::= Rprep Cobject.
ParClause ::= ',' RelClause ','
App ::= ',' {a | an} Cobject ','
```

```
R ::= VERB | Rpas | Radv
Ract ::= VERB {PP}∗ {Rprep}
Rpas ::= is VERBppp {PP}∗ by
Rpronoun ::= {that | which | who}
Rprep ::= PREPOSITION
CompNoun ::= {NOUN}∗ NOUN
Poss ::= {NOUN | CompoundNoun}'s
```

Syntax explanation - In this natural logic grammar, terms in capital letters (i.e. VERB) represent lexicon entries, words with green color (i.e that) represent key words taken from natural language while other ones (i.e. Cterm) represent other elements of the grammar. The symbol * represents a zero to many relation, curly brackets represent elements which are not mandatory, while | represents a logical disjunction.

4.4.2 Simplifications and features handled

Focus on transitivity - Since Natural Logics aims at working over concept graph representations, its grammar focuses on transitive relations which are linking concepts together. Propositions, relative clauses and prepositional phrases are then particularly important. However, intransitive relations are not considered for now. Even if a proposition like *betacell produce* could actually be turned into *betacell produce something*, where the term *something* would refer to the top ontology concept (a concept from which all other concepts are subclasses), some further adjustments would be needed. These changes would involve to discuss how these relations can be integrated in the graph representation and the search approach without creating shortcuts. Similarly, copular verbs without predicates are not considered by the grammar.

Simplifications - In order not to overcomplexify Natural Logics grammar, some restrictions have also been defined. For instance, it isn't possible for now to handle many objects in RC and PP. A concept like *cell that produce insulin and glucose* should then be represented by distributing the RC as *cell that produce insulin and that produce glucose*. Similarly, in order to deal with conjunctions or disjunctions of more than 3 elements (and assuming a distributive reading), the proposition has to be split in many ones. Finally, another restriction is made on isa-relative clauses that are not handled and are deleted in favor of appositions or parenthetical clauses.

Some changes are also motivated by the wish to disambiguate as much as possible the grammar by easing the parsing. For instance, considering passive forms where the object is defined as an object and not as one prepositional phrase modifying the verb actually helps the parsing of passive sentences. Not using any intransitive verbs is also reducing the number of ambiguities. For example, *the doctor listen with a stethoscope the heart* would be translated into Natural Logics as *doctor listen with stethoscope heart*. However, this could be understood as being a transitive verbs used without object and modified by the PP *with stethoscope heart* where *stethoscope heart* is a compound noun [15].

[15] It can be noticed, that this sentence wasn't ambiguous in natural language. This example states on of the bad sides of dropping determiners which actually play a role in natural language

Missing features - Finally, conditions which were not discussed and were built on top of all other extensions are not considered for now.

4.5 Definite Clause Grammar for extended Core Natural Logics

This natural logic grammar theoretically discusses proposition syntax, but cannot be translated as such into DCG clauses. Elements as the "one to many optional blocs" do not have an equivalent in Prolog and recursive rules have then been used.

```
% Propositions with plural formation.
p([p(NP,VP)]) --> np(NP), vp(VP), {VP\=[_,_]}.
p([p(N1,VP),p(N2,VP)]) --> np(N1), [and], np(N2), vp(VP), {VP\=[_,_]}.
p([p(NP,V1),p(NP,V2)]) --> np(NP), vp(VP), {VP=[V1,V2]}.
p([p(N1,V1),p(N1,V2),
   p(N2,V1),p(N2,V2)]) --> np(N1), [and], np(N2), vp(VP), {VP=[V1,V2]}.

vp(vp(V,pred(A))) --> rterm(V,_,copular), adjs(A).
vp(vp(V,N)) --> rterm(V,trans,_), np(N).
vp([vp(V,N1),vp(V,N2)]) --> rterm(V,trans,_), np(N1), [and], np(N2).
vp(vp(V,N)) --> rterm(V,trans,_), np(N1), [or], np(N2), !,
                                          {np(N1,S1,[]), np(N2,S2,[]), !, isa(S1,N), isa(S2,N)},
                                              {print_common(N,S1,S2)}.

% Adjectives as predicate.
adjs([A]) --> adj(A,predi). % Always an adjective at least.
adjs([A|T])  --> adj(A,predi), adjs(T).

% Concept nouns accepting modifiers.
np(np(N,mod([]),ext([]))) --> n(N). % Kept for computational reasons.
np(np(N,mod(M),ext(E))) --> pre(Q), n(N), post(P,E), {append(Q,P,S), sort(S,M)}.

% Defining pre-modifiers.
pre(T) --> pre1(T). % Case without possessive.
pre([G|T]) --> cn(N1), ["s"], pre1(T), {G = ger([affiliation],N1)}.
pre1([adj(A)|T]) --> adj(A,_), pre1(T).
pre1(T) --> pre2(T).
pre2([cn(N1)]) --> n(N1). % Not needed, for efficiency reasons.
pre2([cn(N1)|T]) --> n(N1), pre2(T).
pre2([]) --> {true}. % Potentially no pre-modifiers.

% To allow compound nouns in possessives.
cn(np(N)) --> n(N).
cn(np(N, mod(M))) --> pre1(M), n(N).

% Defining post-modifiers.
post(M,[]) --> post1(M). % Case without apposition.
post(M,[A]) --> app(A), post1(M).
post1([pp(P,N)]) --> prep(P), np(N).
post1([rc(V,N)]) --> [that], rterm(V,trans,_), np(N).
post1([pp(P,N)|M]) --> prep(P), np(N), align(_), post1(M).
```

disambiguation.

```
post1([pp(P,N)|M]) --> prep(P), np(N), post1(M).
post1([rc(V,N)|M]) --> [that], rterm(V,trans,_), np(N), align(_), post1(M).
post1([rc(V,N)|M]) --> [that], rterm(V,trans,_), np(N), post1(M).
post1([]) --> {true}. % Potentially no post-modifiers.

% Defining appositions.
app(ap(N)) --> [","], [a], np(N), [","].
app(ap(N)) --> [","], [an], np(N), [","].

% Defining parenthetical clauses.
app(pc(V,N)) --> [","], [which], rterm(V,trans,_), np(N), [","].

% Relation terms
rterm(verb(V,mod(M)),X,T) --> rterm1(V,X,T), advs(P), pps(Q), {append(P,Q,M)}.
rterm(verb(V),X,T) --> [is], [V], {lex(_, X, T, V)}, [by]. % Passive trans.
rterm(verb(V,mod(M)),X,T) --> [is], [V], {lex(_, X, T, V)}, pps(M), [by].

rterm1(active(isa),X,T) --> [isa], {T = copular, X = trans}.
rterm1(active(V),X,T) --> [V], {lex(V, X, T, _)}. % Active transitive.
rterm1(active(V,P),X,T) --> [V], {lex(V, X, T, _)}, prep(P). % Act intrans with prep.

% Adverbial PPs.
pps([pp(P,N)]) --> prep(P), np(N).
pps([pp(P,N)|T]) --> prep(P), np(N), align(_), pps(T).
pps([pp(P,N)|T]) --> prep(P), np(N), pps(T).
pps([])  --> {true}. % Potentially no adverbial PPs.

% Adverbs.
advs([A|T]) --> adv(A), advs(T).
advs([]) --> {true}. % Potentially no adverb

% Alignment keywords.
align(X) --> [X], {X = and}.
align(X) --> [X], {X = ","}.

% Access to lexicon entries
prep(P) --> [P], {lex(P, preposition)}.
n(n(N)) --> [N], {lex(N, noun)}.
adj(A,T) --> [A], {lex(A, adj, _, T)}.
adv(A)  --> [A], {lex(A, adv)}.
```

This grammar completely handle the set of features described previously. The parser has also been implemented in a way that would ease the decomposition as graph. In fact, it returns each noun phrase with a list of its modifiers (having a restrictive value) and a list of its extensions. Even if it does not ease the reading by turning some simple nouns like *np(pancreas)* into *np(n(pancreas), mod([ ]), ext([ ]))*, this approach is valuable for complex concept terms such as *glucose production by pancreas* since it brings out more explicitly the semantic structure: *np(n(production), mod([cn(n(glucose)), pp(by, np(n(pancreas), mod([ ]), ext([ ])))]), ext([ ]))*.

4.6 Differences with Description Logics

After having defined a DCG parser for Natural Logics and its extension features, a graph representation for its concepts, and some basic inference rules, this section summarizes some of the main differences between Natural Logics and Description Logics.

In fact, Description Logics and Natural Logics both arises from the will to represent ontology structures in a more efficient and readable way than FOL while keeping a relatively broad expressivity. Both get rids of explicit variable representation, but substantial differences in their approach exists.

4.6.1 Differences in the approach

Concept representation - Description Logics has a clear distinction between individuals, classes and roles. Individuals refer to axiomatic definitions while Natural Logics, consider a definition of concepts which covers both classes and individuals while transitive verbs represent relations between concepts. No explicit distinction is made between instances and classes in Natural Logics.

Even if some similarities can be seen between both concept representations, especially with modifiers as for RC (*cell that produce insulin*) that can be expressed as intersection with the existential quantifier ($cell \sqcap \exists produce.insulin$), Description Logics deals with a larger definition of concepts (negation of concepts, disjunction of concepts) so that some cannot be translated directly in Natural Logics as such, while Natural Logics representation is definitely closer to natural language.

Negation and existence - Description Logics is based on the Open World Assumption and consider the existence of empty concepts, meaning concepts that potentially don't have any instances. From these two properties arises also the need to define explicitly all concepts disjunctions in order to define them as not overlapping. This is constraints-full, but allow to deal with negation. On the other hand, Natural Logics uses the Closed World Assumptions and does not consider empty concept classes. Negation arises then from the failure of a demonstration, but can also be emulated through the underlying ontology defined for the domain specific model[16].

Resolution approach - Finally, another main differences between DL and NL relies in the resolution approach chosen. While DL refers to a set of complex inferences rules, NL focuses on basic inference rules mirroring the human reason, on concept graph representation and on graph search.

4.6.2 Advantages of Natural Logics

Even if Description Logics has the advantage of being a longer approach with a larger community, of possessing a set of distinctions enabling the choice of different complexity and expressivity, Extended Core Natural Logics overcomes the limitations

[16]This aspect of Natural Logics is discussed in the next session.

of DL defined previously (c.f. 2.3.2).

Natural language syntax - Since Natural Logics aims to use natural language as a vessel to infer knowledge, its syntax is closer to natural language. Its translation from and to natural language is easier to process. Natural Logics is also easier to read and to understand, so that it can be used also by non expert user.

Representing structures - While it appears as difficult to handle the four different syntactic structures defined from Aristotelian Logics underlying Natural language sentences in DL [30]. Natural Logics can be extended easily in order to represent all of them. Especially the main cases of $\forall\exists$ and $\exists\forall$ are both already implemented through the default proposition structure and the passivization process of sentences.

Dealing with adverbials - Finally, even if Description Logics defines a pretty complete model to deal with relations including composition, inclusion, intersection or inverse relations, Description Logics fails at dealing with adverbials modifying relations. While not considering any operation on relations in order to keep simple inference rules, Natural Logics successfully handle event-oriented adverbials based on nominalization process. In Natural Logics the inversion of relations is handled through passivization done during the result rendering of the graph search (see chapter 7).

4.7 Ambiguity problem in Extended Core Natural Logics' grammar

The last sections tried to avoid ambiguities in Natural Logics' grammar, but at least two ambiguous cases can still be noticed.

Example 1 - The first case is related to nested prepositional terms and relative clauses. For instance, the proposition C that R1 C1 in C2 in C3 and that R2 C4 can be interpreted in two ways:

- C that R1 (C1 in (C2 in C3 and that R2 C4))
- C that R1 (C1 in (C2 in C3) and that R2 C4)

In this example, the reference of a relative clause is ambiguous, since the first interpretation states that the relative clause R2 C4 applies on the concept C2 while the second one states that the relative clause applies on the concept C1.

Example 2 - The second case of ambiguity is due to the use of the complex nouns and adverbials. For instance, the sentence C that R in C1 C2 in C3 C4 can be interpreted in two ways:

- C that (R in C1) (C2 in C3-C4) $\rightarrow$ C that R in C1 C' with C' = C2 in C3-C4
- C that R in (C1-C2 in C3) C4 $\rightarrow$ C that R in C" C4 with C" = C1-C2 in C4

The first interpretation identifies C as the subject, R in C1 as the relation and C2 in C3-C4, where C3-C4 is a compound noun, as the object. In the second case, the relation is identified as R in C1-C2 in C3, where C1-C2 is a compound noun, and C4 is identified as the object.

This ambiguity type is present when dealing with the translation into Natural Logics of the following sentences of the insulin article [39], (Appendix A & B):

- betacell, secrete, at, high, glucose, concentration, into, blood, insulin.
- betacell, stop, at, low, glucose, concentration, insulin, secretion

A solution in order to disambiguate Natural Logics grammar is then needed.

CHAPTER 5
Disambiguating Extended Core Natural Logics

The last section defined a set of extensions for Natural Logics in order to cope with most syntactic structures from scientific texts. However, these features come along with an ambiguity problem that is inherent to all languages. After defining the scope of ambiguity types for Natural Logics and especially its resolution in natural language, this section proposes an approach to disambiguate sentences in Natural Logics, based on ontological and semantic analysis of its elements.

This section discusses the application of this approach based on the first paragraphs of the Wikipedia article on *Insulin* [39]. This section aims to explain the approach that should be used and not to define a complete model working over all potential cases that Natural Logics would face in the future. Further extensions of the model would be needed to cover more topics.

5.1 Inherent ambiguity problems in natural language

One morning, I shot an elephant in my pajamas...
How he got into my pajamas, I don't know.

Groucho Marx, in Animal Crackers, 1930

This sentence shows an example of PP-attachment ambiguity, serving here a comic effect.

5.1.1 Ambiguity and its resolution in natural language

Even if it seems to be done effortlessly by the human brain, dealing with ambiguities remains one of the hardest problems that language processing applications have to deal with.

5.1.1.1 Global and local ambiguity

Two types of ambiguity can be defined depending on their resolution during the sentence processing. On the one hand, a sentence is characterized as globally ambiguous if it has at least two possible interpretations [22]. The sentence: *Someone shot the servant* *of the actress who was on the balcony* shows a global ambiguity since it is impossible to know whether the actress or the servant was on the balcony. On the other hand, a sentence is said to be locally ambiguous if the ambiguity encountered is resolved during the processing. Locally ambiguous sentence have then only one possible interpretation, but often contains word sequences whose interpretation appears as unusual. This arises because the human brain processes words separately and makes assumptions on the structure of sentence before its end. *Garden Path Sentences* are a well-known type of sentences containing local ambiguities. They start in a way that the reader's interpretation will most likely be incorrect. For instance, the sentence *The old man the boat*, meaning *The old people are serving the boat*, will probably be at first processed considering that *The old man* is a Noun Phrase. However, to be grammatically correct, it should be understood as a noun followed by a verb.

5.1.1.2 Semantic, structural and lexical ambiguity

While global and local ambiguities are distinguished based on their resolution, at least six types of ambiguities can be defined based on their origin [13].

Lexical ambiguity - Lexical ambiguity refers to the polysemy of words and the potential ambiguity they can generate. For instance, the sentence *I saw a bat* can be understood in two way since *saw* can refer as well to the past of *see* as to the present of *saw* (cutting with a saw).

Structural ambiguity - Structural or syntactic ambiguity are related to the structure of the sentence. The example from Groucho Marx with the elephant quoted previously belongs to this category.

Semantic ambiguity - Semantic ambiguity arises from the possibility to give different meaning to some words and sentences even after resolving structural and lexical ambiguities. For example, in the sentence *The dog is chasing the cat*, the word *dog* refers to a specific dog while in the sentence *the dog has been domesticated for 10,000 years* refers to the whole species.

Anaphorical ambiguity - Anaphorical ambiguities usually refer to ambiguous use of pronouns in sentences. The Winograd Schema challenge[1] [14] is based on this type of ambiguities and request competitors to process couples of sentences like: *The father couldn't lift his son because <u>he</u> is too (heavy/weak)* and to disambiguate in each case the referent of the ambiguous pronoun.

Figurative ambiguity - This type of ambiguity arises from the use of speech figures such as metonymies or metaphors.

[1]The Winograd Challenge has been created as a alternative to the Turing test which was criticized for its subjectivity and for its relation to the creation of a personality for the software being tested. The Winograd Challenge is based on a set of schema that should be solved by the candidate software and whose success rate is then objective.

Ellipsis ambiguity - In speeches, it is common to omit some words that would be required to obtain a grammatical sentence. For instance the sentence: "I am allergic to tomatoes. Also fish." is usually understood as "I am also allergic to fish" rather than "Also, fish are allergic to tomatoes.", even if the latter construction would seem more grammatical. This type of ambiguity is less common in written language.

5.1.1.3 Disambiguation strategies for human brain

Even if for now if it is still unclear how the brain exactly processes these ambiguities, it is quite sure that it involves a prosody analysis [2] during oral discussions [17], but also involves background knowledge and a world comprehension [33].

Different models have then been proposed based on probabilistic rules (Probabilistic models) or by miming humans that focus on the most important information when facing complex sentences (Good Enough Theory). Some rules, named Frazier's principles [16], are often considered as default disambiguation rules:

- Late closure principle: assumption that new clauses tend to be associated with the current phrase or clause being processed. For instance, in the sentence *John said he would leave yesterday"*, the adverb *yesterday* would refer to *leave* while in the sentence *John said yesterday he would leave"*, it would be associated with *say.*
- Minimal attachment: observation that listeners and readers initially tend to process sentences as the simplest syntactic structure for the input they have at the moment. This observation also explains the misinterpretation of garden path sentences.

From these principles, the late closure should be kept when dealing with ambiguous structures. As a consequence, when encountering a noun phrase such as *betacell in pancreas that produce insulin*, the production of insulin should be understood as referring to *pancreas* and not to *betacell*[3].

5.1.2 Ambiguity scope for Extended Natural Logics

Not all of the ambiguity types presented previously are considered relevant for Natural Logics. Lexical ambiguity is not considered here since the domain of discourse

[2]Prosody brings together all elements of speech that are not related to pure phonetic such as intonation, tone, stress, breaks, emphasis on some syllable or accents.

[3]If the intended meaning is to refer to the *betacell*, it would be expected that the sentence would be ordered as *betacell that produce insulin in pancreas* since the relation between insulin located in pancreas appear less likely than the relation between production and a location, or by using the keyword *and* to explicitly define the alignment of the relative clause: *betacell in pancreas and that produce insulin.* It can be noted that an alignment keyword isn't used when the referent is semantically obvious. For instance *production in pancreas by glucogenesis* is actually an aligned structure since its meaning is *production in pancreas and by glucogenesis.* The explicit alignment is removed here since the PP-by cannot refer to *pancreas.*

(bio-medical scientific texts) is relatively narrow. Semantic ambiguities are also not considered since bio-scientific texts usually discuss sentences as general rules. For instance, a sentence like *the absorbed glucose is converted into glycogen* refers to glucose in general and not to a specific instance. This interpretation is motivated by the underlying quantification which is $\forall x, absorbed(x) \land glucose(x) \rightarrow \exists y(converted(x, y) \land glycogen(y))$ rather than $\exists x, absorbed(x) \land glucose(x) \land (\exists y converted(x, y) \land glycogen(y))$. Concerning, anaphoric ambiguity, this remains a really hard problem that for now can be considered as a pre-processing issue. Finally, figurative and ellipsis ambiguities are not discussed since their use is limited in scientific texts which aims at clarity.

As a consequence, Natural Logics focuses for now only on solving structural global ambiguities. However, other types of ambiguity should be kept in mind especially if the domain of discourse is extended. Local ambiguities are not considered at all here due to the use of parser finding all and only the grammatically correct structures (even if unusual).[4]

5.2 Disambiguation based on ontological and semantic analysis

As an obvious solution, the use of brackets can be considered to explicitly delimit the scope of each element. This has the advantage of being easy to understand and to implement inside the DCG clauses. However, this does not define a viable solution as such since it implies either some human interaction to disambiguate the sentences or the design of an automated process.

5.2.1 Disambiguation approach

A more elegant approach is to determine which of the generated parse-trees is the correct one based on semantic analysis. In fact, the Natural Logics model described previously enables to detect syntactically correct sentences, but do not consider semantic correctness. As a consequence, phrases such as 1.b) and 2.b) could be represented even if they don't make sense.

```
1.a) the doctor observes the patient
1.b) *the table observes the patient
2.a) an autoimmune disease
2.b) *an autoimmune doctor
```

The first step toward disambiguation is then to eliminate expressions which are well-formed syntactically, but ill-formed semantically [7]. This can be achieved by imposing restrictions on the subject and objects of relations and on the referent that adjectives modify. The main idea is to rely on an ontological classification of all concepts of the

[4]It is important to note that whatever solution is chosen to solve this ambiguity problem, it won't have much consequences on the graph generation and graph search algorithms. In fact, the solution presented will be applied so that the parse tree structure would remain identical.

domain of discourse of Natural Logics in order to select types on which adjectives and verbs could be applied. For instance, a distinction between *animate entities*, such as a doctor, and *inanimate entities*, such as a table, could be used to define a constraint on the subject of the verb *observe* which implies a disposition for sensory perceptions of its subject.

The second step of the presented approach focuses on the disambiguation of sentences and NPs containing one or more PPs such as 3.a), 3,b) and 4) . It relies on the existence of an ontology of concept mentioned previously and on the definition of a finite set of semantic roles for prepositional phrases.

```
3.a) the doctor observes the patient with a stethoscope
3.b) the doctor observes the patient with diabetes
4. the production of glucose by the liver
```

Based on the ontological classification of concepts and their potential relations, it would then be possible to define affinities between concept types and preposition use [25]. Similarly, the same approach based on affinities is discussed in order to deal with compound noun constructions and with possessive forms. Finally, if these steps reveal to be insufficient, and the situation remains ambiguous, Frazier principles are used.

5.2.2 Defining an ontology skeleton for bio-scientific texts

The definition of ontologies is a classical problem in computer science and especially in bio-medical sciences. They often result from an empirical representation of their author's view of the world, the entities it is composed of, their properties and their relations. It is then quite impossible to reach a complete solution or a global agreement of the whole scientific community. A lot of ontologies have been created for each topic (genes, diseases, medical investigations...), but they are usually hard to merge together and they don't always reflect the need of the user. Cross domain communication and adaptation are usually therefore challenging.

As a consequence, the chosen approach relies on the use of already existing top order ontology which is extended with concrete instances of the domain of discourse[5].

5.2.2.1 Working over an upper ontology of concepts

An upper ontology, also named top ontology, is a small ontology of concepts designed in order to support integration in scientific research. Their ambition is to be used as a common root for domain specific ontologies. As a consequence, they focus on defining domain and language independent conceptualizations of the world. The definition of upper ontologies goes back until Aristotle's ontology and is still a current research area

[5]The objective here is then not to define a new ontology aiming for completeness, but to use top order formal ontology as basis a for good practice and then create a domain specific ontology that can be used to demonstrate the benefits of the disambiguation approach.

in computer science and especially in AI [21]. Among the upper ontologies available at the net, a deeper analysis has been made on the General Formal ontology (GFO) [23] and especially on Basic Formal Ontology (BFO) [37].

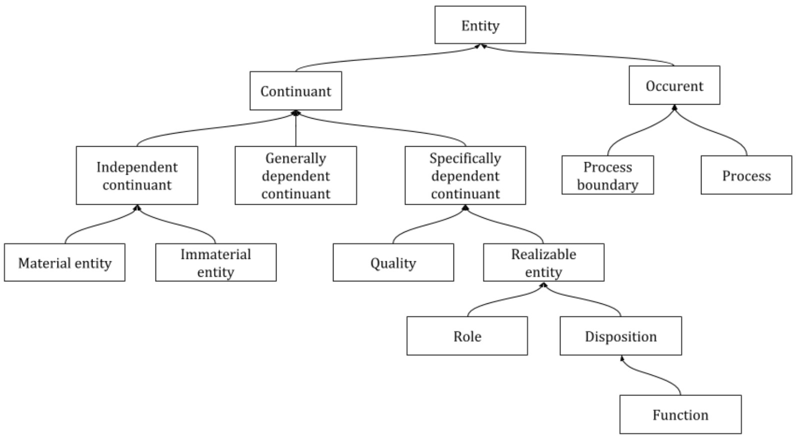


Figure 5.1: Upper Ontology from BFO.

Main distinctions - The first distinction between world entities is based on their relation to time, leading to the opposition between *continuants* (also called *persistents*) and *occurents*. *Continuants* refer to entities which do not really variate with time. For instance, humans are an example of concrete continuants. Even if they may change slightly during time, each person remains the same individual all his life and thus, at many different points in time. On the other side, *occurents* refer to entities that occur and are thus highly time dependent like processes.

Occurents - *Occurents* highlight a distinction between processes themselves and their boundaries, which are more time delimited.

Continuants - Among *continuants*, the main distinction is based on the condition of their existence. As a consequence, *independent continuants* and *dependent continuants* can be distinguished. On the one hand, *independent continuants* exist without any conditions, they cover both concrete (human, organs, furniture) and abstract continuants (ideas). On the other hand, *dependent continuants* depend on the existence of an *independent continuant* to exist. They are separated in two categories: *specifically dependent continuants*, (like qualities, functions, roles or dispositions), which disappear when their bearer disappear, and *generally dependent continuants* which are characterised by their copiability (information, pdf-files or gene sequences) and thus, might not disappear when their bearer does.

Among *specifically dependent continuant*, there exist a distinction between *qualities* (such as colors, properties) and *realizable entities* such as *roles* or *dispositions* [6]. On the one hand, a *role* is a *realizable entity* which exists because the bearer has some physical, social or institutional capacity and that can turn into a *realizable event* (an *occurent*). For instance, being a doctor is a *role* beared by people. When retiring, the role of doctor disappear, but the bearer remains the same. This is especially clever when considering evolution of roles of concepts through time. A doctor has not always been a doctor, but was first a child, then a student and finally a doctor without never losing its own identity. On the other hand, a *disposition* is a *realizable entity* which is specifically based on a physical capacity. If this *disposition* ceases, it necessarily implies a physical changes of the bearer, in opposition to the lost of the *role*. For instance, a hand has a *disposition* to grasp objects. If a hand cannot grasp objects, then it implies a physical disability and thus a physical change of the hand. Similarly, a *function* is a type of *disposition* that exists based on a physical capacity of the bearer that he possesses in order to realize a specific process through being, evolution (for biological entities: the purpose of betacells is to produce insulin) or through intentional design (for artifacts: the function of a table is to hold)[36].

Another important design taken from BFO is the consideration of a canonical anatomy model in the ontology. Differences or malformations are then considered as specific examples and are not integrated in the base-model of anatomic parts. It should be noted that the ontology constructed here is not purely hierarchical since some elements might appear at different places. For instance, a liver can be seen as food or as an organ [25].

Simplifications - However, the approach of upper ontology is often really abstract, almost philosophical, and then not all distinctions need to be considered in practice. For instance, it is more interesting to consider events and states of concepts (their presence or lack) instead of process boundaries. Moreover, since Natural Logics does not consider time extensions, some elements like roles are not needed. It is then simpler to reduce a role as *Doctor* to its bearer and thus, to declare Doctor as a subtype of *human* and not as a *role* (figure 5.2). The motivation is that in most cases, the ontological restrictions that can be applied refers to the bearer and not to the role. For example, in *the doctor observes the disease*, the doctor role is accepted as agent due to its underlying bearer.

5.2.2.2 Tailoring the ontology to the considered domain

After considering these simplifications, some changes were also made to tailor this ontology to the needs of the domain of discourse using the insulin article as a basis [39].

Continuants - The distinction made between *material* and *immaterial entities* is kept in order to distinguish the ones with a physical presence from abstract ones that don't. In order to distinguish *material entities* that have a sensory perception and those that have not, a distinction has been made between *inanimate* and *animate*. Among *inanimates*, a distinction is then made between *countable* and not countable elements,

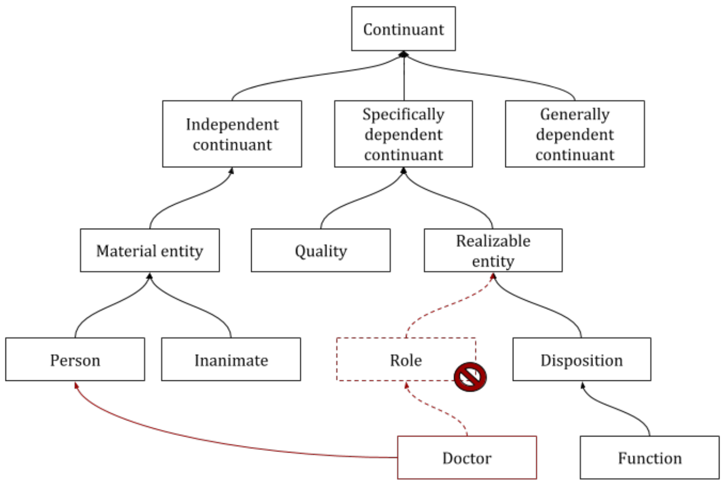


Figure 5.2: Simplification of roles to their underlying bearers.

mass, since it induces changes in the way they are perceived, especially quantitatively.

Occurents - The most interesting *occurents* for Natural Logics are *processes* and *states*. *States* are distinguished into *presence* and *lack*. *Processes* are split between *creative processes* and *non creative processes* depending on their ability to produce something or not. Among *creative processes*, a distinction is then made between *transformations* and *creations*. Among *non-creative processes*, an important class refers to *enhancement processes* which represents processes that affect other processes such as *inhibition* or *promotion*.

5.2.3 Defining a set of semantic relations across the ontology

The next step aims at defining a finite set of binary role relations between concepts of the ontology. Some of these relations are defined in the BFO and are classical across ontologies (table 5.1).

Ontology relation	**Definition**
X isa Y	entity it is included in
X beared_by Y	entity holding a property or hosting a process
X bearer Y	the one that holds the property
X part of Y	entity it belongs physically to
X has part Y	entity it is composed of

Table 5.1: Ontological relations.

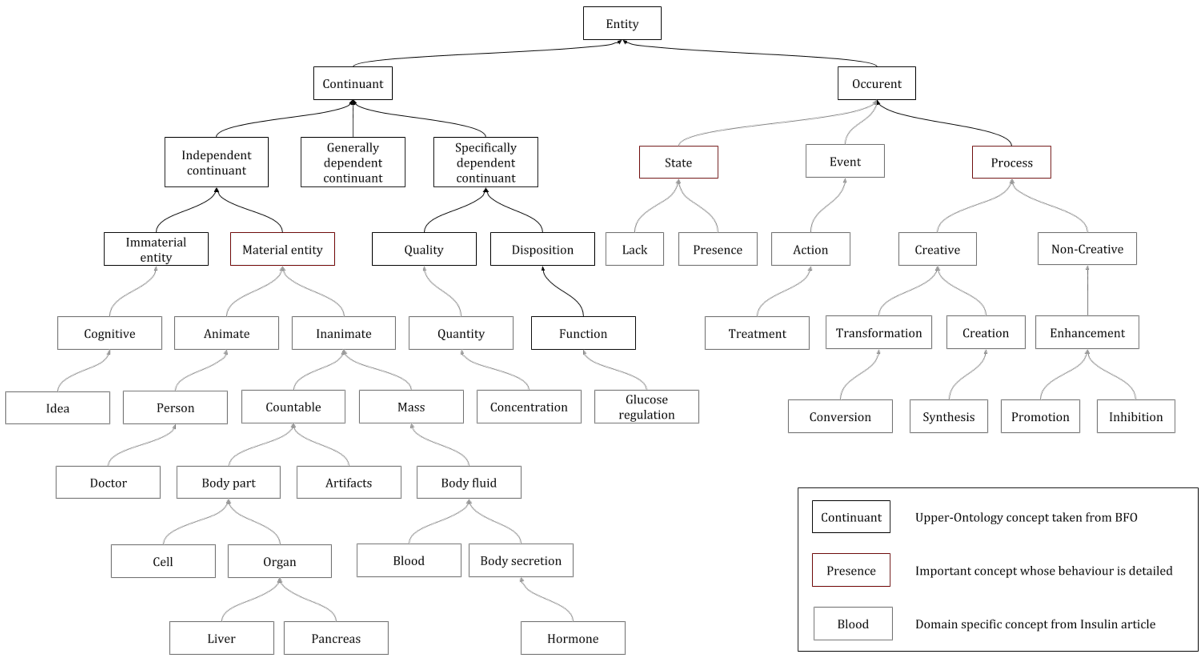


Figure 5.3: Part of the domain Specific Ontology for Natural Logics.

The other ones are based on semantic relations, also named thematic relations (table 5.2). The definition of a finite set of relations is always a complex topic and hasn't reached an agreement yet among the linguistic research community.

The set mentioned here is based on the one available on wikipedia [42] and has then been modified in order to cope with the relations present in the insulin article [39]. This set aims particularly at defining general relations and might not consider some smaller distinctions. For instance, the *agent* is considered as the entity that performs the action, while some linguists consider it as an entity that deliberately performs the action, in opposition to an *experiencer* who receives a sensory input (like in, I cried) or a *force* which mindlessly performs the action (like a cell producing something). Similarly, an entity that undergoes the action without changes its state (*theme*) and an entity that undergoes the action and changes its states (*patient*) are brought together under the term *patient*. However, the distinction between *patient* and *result* is kept since the domain of discourse contains a lot of creation processes.

5.2.4 Ontological analysis of important concepts

Based on this set of semantic relations, it is then possible to define models for the main concepts of the ontology in order to understand how they are related to each other.

Thematic relation	Definition
agent	entity that performs the action
patient	entity that undergoes the action
result	entity that is the result of the action
instrument	entity that is used to carry the action
accompanient	accompanies the action without influencing it
location	where the action occurs
source / origin	where the action started
direction / goal	where the action is directed towards
time	time at which the action occurs
manner	the way in which the action occurs
purpose	the reason why the action is performed
cause	what caused the action in the first place
affiliation	what the action is linked to
characterization	entity characterizing the action
condition	condition for the action to happen

Table 5.2: Thematic relations for Natural Logics.

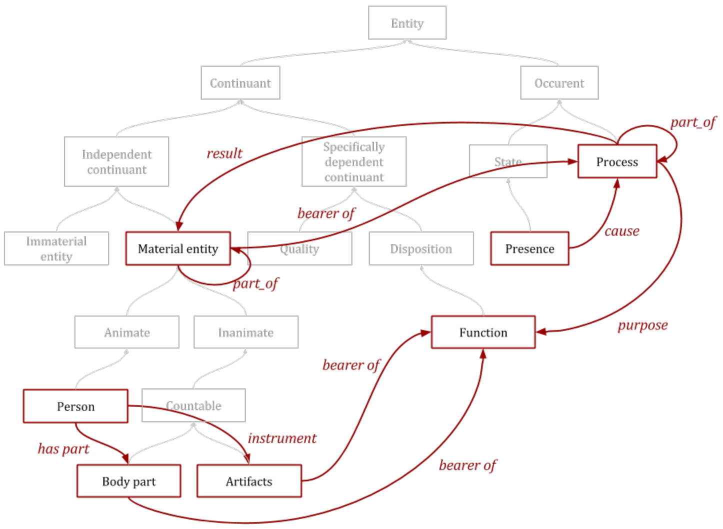


Figure 5.4: Example of relations across major concepts.

Among them, a specific attention is given to *processes* (table 5.3), to *materials* (table 5.4) and to *states* (table 5.2.4) since they are the most common entity type.

While *processes* appear to be involved in a lot of potential semantic relations, the ontological structure around *materials* seems to be simpler. *Materials* generally have a location or an origin, some of them can have other relations such as *body parts* and

Relation	**Ontology object**	**Example**
agent	material	production by pancreas
patient	material	conversion of glucose
result	material	creation of glucose
instrument	material, process	production with tools, production by synthesis
location	material	production in pancreas
source / origin	material	production from blood
direction / goal	material	production into pancreas
manner	quality	production in urgency
purpose	function	production for sustainment
cause	presence of ind. cont.	production caused by glycogen (presence)
isa	occurent	
depends on	realizable entity	
beared_by	material entities	
bearer	material entities	
part of	processes	
has part	processes	

Table 5.3: Ontology types for referents of relations around processes.

Relation	**Ontology object**	**Example**
location	material	cell in pancreas
source / origin	material	glucose from the blood
isa	dependent continuant	
part of	material	
has part	material	

Table 5.4: Ontology types for referents of relations around materials.

artifacts which have a purpose (ontologically referring to a function), like in *a tool for observations.* However, a lot of relations cannot be considered for *materials* such as direction. Especially if a phrase like *glucose into the blood* can be found, the PP-into is actually linked to the verb of the sentence, like in *produce glucose into the blood.*

States can combine with a variety of types. Any kind of entity can be the patient of a *state. States* have a location, but no origin (they might have a cause though) or direction or purpose. The cause of a *state* can be another *state* (the lack of medication causing the presence of diseases), a *process* or an *independent continuant*[6].

[6]From a strict point of view, the cause of a state cannot be a independent continuant. If a man started a fire, then he is the initiate and, in the broad sense, the source of the presence of the fire. However, the fire didn't start because of the man being himself, but due to its behaviour or actions (*processes*). A metonymy is then involved to relate a process to the continuant performing it. However, as a simplification to handle this case, independent continuant can be considered as cause of states

Relation	**Ontology object**	**Example**
patient	entity	presence of insulin (material) presence of red (quality) presence of a disposition (disposition) presence of synthesis (process)
location	ind. continuant	presence in pancreas presence in my mind
manner	quality	fortunate presence
cause	state, process	presence caused by the presence of insulin presence caused by the synthesis of hormone
condition	quality	presence at high insulin concentration
isa	occurent	
depends on	entity	

Table 5.5: Ontology types for referents of relations around states.

5.2.5 Defining constraints on the parser

5.2.5.1 Defining constraints on relation terms and adjectives

Based on the ontology defined previously, the idea is now to define types to the relations and adjectives that can be encountered in the domain of discourse [7] (table 5.6). For instance, it can be defined that the verb transitive verb produce can be used with creation *processes* as subjects and with body fluids as objects. Then, all the subtypes of the subject and the object can be accepted respectively as subject and object.

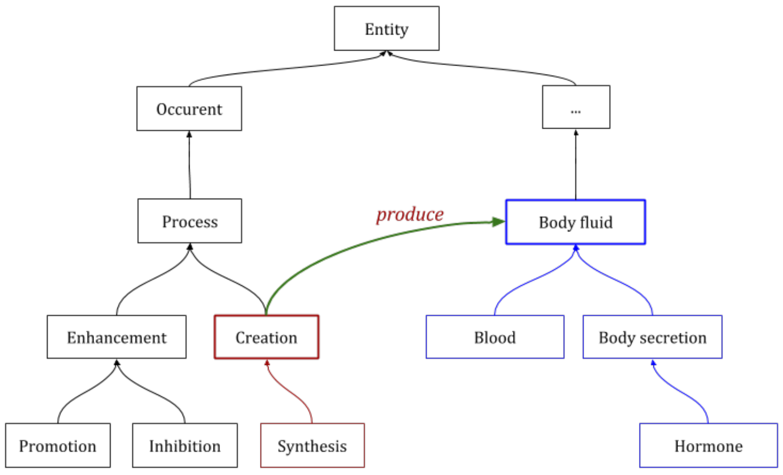


Figure 5.5: Scope of subjects and objects for the transitive verb *produce*.

Relation	**Type**	**Subject**	**Object**
produce	transitive	creation, transformation, organ, cell	body fluid
observe	transitive	animate	entity
inhibit	transitive	state, enhancement	process
smile	intransitive	animate	∅

Table 5.6: Examples of constraints applied on verbs.

Similarly, the same restrictions can be defined on adjectives as in the table 5.7.

Adjective	**Referent type**	**Example**
high	quantity countable	high concentration high castle
small	countable animate	small table small cat *small glucose
opposite	spe. dependent	opposite color (quality) opposite function (function) *opposite doctor
pancreatic	body part	pancreatic cell *pancreatic doctor
sensitive	entities	sensitive cell sensitive process

Table 5.7: Examples of constraints applied on adjectives.

5.2.5.2 Defining affinities between concepts

Prepositional phrases - Based on the insulin article [39] the set of the most common prepositions has been defined and an analysis of the concepts they were relating as been performed. The results can be found in the table 5.8.

This can be refined to some specific subtypes to avoid some awkward constructions (like *insulin in the table*, a body substances in a piece of furniture). The challenge is to find the right balance between something that is too generic (and would then allow incorrect relations) and something that is too specific (and would discard some correct relations).

Compound nouns - Affinities for compound noun constructions aim to cope with the schema driven approach defined previously. Since this approach usually involves

Preposition	**C**	**D**	**Semantic relation**
in	material	material	location
in	process	material	location
in	quality	material	beared_by
in	information	process	affiliation
by	process	material	agent
by	process	process	manner
by	process	artifact	instrument
into	transformation	material	result
into	creation	material	direction
of	creation	material	result
of	transformation	material	patient
of	process	material	beared_by
of	process	process	has_part

Table 5.8: Examples of affinities of prepositions: C-PREP-D.

to deal with metonymies, this approach is restricted to *processes*, especially *creative processes*, for now (table 5.9).

Semantic relation	**C**	**D**
agent	cell	process
agent	organ	process
body_fluid	process	result

Table 5.9: Examples of affinities for compound nouns: C D.

Based on this, a term like *glucose production* would be recognized as semantically correct with a patient relation while a compound noun as *synthesis production* would be discarded.

This approach can be extended to adverbial PPs by considering the ontological type of the nominalization of the verb to especially distinguish state verbs and event verbs.

5.2.5.3 Further extensions

Improving compound noun comprehension - The model discussed around compound nouns could be also extended based on the frequently used structures. In fact, some affinities are stronger than others and are then more frequent in natural language. For instance, the affinity between a *creative process* and its *result* is stronger than with its *agent*. Thus, a compound noun like *liver production* where *liver* is the *agent* wouldn't usually be mentioned, unless the *result* has already been mentioned in a previous compound noun construction. Finally, the order of the nouns in compound nouns could also be considered. In fact, it is more common, at least for *creative processes*, to find an *agent-result-process* ordering like in *liver glucose production* than a *result-agent-process* ordering which results in the somehow awkward construction *glucose liver production.*

Handling possessives - This approach might potentially be extended later on to cope with possessive forms (distinguishing affiliation from creation and possession) and with prepositional verbs (like *reside in* which induces a location relation). It could especially be used to carry out explicitly the underlying semantic relation in a similar way as discussed previously.

5.2.6 Implementation

The implementation of the constraints in the parser is available in the appendix C.2.1. At first, a solution based on the definition of transitive rules for constraints and affinities had been chosen, but it turned out to be computationally inefficient. The implementation was finally done by checking that the concepts are subclasses of the types defined as constraints using a variation of the *isa* predicate. One of the main challenges of the implementation has been to deal with the constraints on pre-modifiers. In fact, they are processed by the parser before the noun they refer to, it was then harder to verify if the constraint holds.

5.2.7 Examples of disambiguation

This section provides a few examples where the disambiguation approach works successfully. A detailed analysis of the disambiguation results on the reference article is available in chapter 8.

5.2.7.1 Removing semantically incorrect sentences

The following examples are inspired from [7] and show the removal of semantically incorrect structures through ontological analysis.

```
?− np(X,Z,[skeletal,cell],[]).
X = np(n(cell), mod([adj(skeletal)]), ext([])),
Z = const(cell) .


?− time(np(X,Z,[skeletal,promotion],[])).
false.

?− p(X,[betacell,produce,insulin],[]).
X = [p(np(n(betacell), mod([]), ext([])),
        vp(verb(active(produce), mod([])), np(n(insulin), mod([]), ext([]))))]

?− p(X,[insulin,produce,betacell],[]).
false.
```

5.2.7.2 Disambiguation of PP referential

This example shows the disambiguation of a noun phrase. The concept has been chosen as a production since the model is centered around them. While the grammar without any ontological analysis proposes 42 different solutions and did not suggest the right parsing tree on the first trial, the ontological parser could.

```
% Number of distinct parse trees found by the parser:
?− setof(X,np(X,[production,in,pancreas,of,glucose,by,betacell,that,produce,insulin,
                        by,glucogenesis],[]),R), length(R,Z).
R = [...]
Z = 42.
```

This parser also returns the type of relation which can be useful in order to label edges on the graph.

```
% Solution and time to return the first solution:
?− time(np(X,Z,[production,in,pancreas,of,glucose,by,betacell,that,produce,insulin,by,glucogenesis],[])).
X = np(production, mod([pp(in, [location], np(pancreas)),
                        pp(of, [result], np(glucose)),
                        pp(by, [agent], np(betacell, mod([rc(verb(produce), np(insulin))]))),
                        pp(by, [manner], np(glucogenesis))])),
Z = const(creation)
```

CHAPTER 6

Graph generation

The previous, section defined an approach to disambiguate propositions in Natural Logics based on ontological analysis of concepts. This section now focuses on extracting knowledge from texts by defining an algorithm able to decompose automatically Natural Logics propositions into their corresponding concept graph representations.

6.1 Approach for knowledge extraction

In order to extract knowledge from a text, each proposition is decomposed individually using a divide-and-conquer approach. Complex propositions are turned into a set of simple propositions which are individually transformed into graphs. Based on the unicity of concepts, identical concepts are merged together.

The main part of the algorithm is then to recursively decompose sentences. This decomposition is done in three consecutive steps: *canonical decomposition*, *application of inference rules* and *application of subsumption rules.*

The picture 6.1 shows the application of these three steps in the generation of the concept graph of the proposition *cell that produce insulin, which isa hormone, regulate body.*

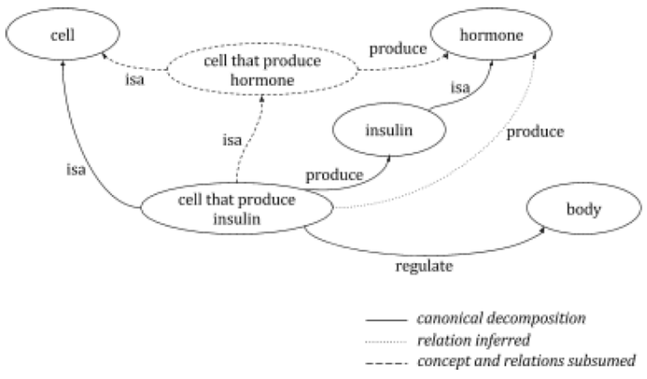


Figure 6.1: Approach for sentence decomposition.

6.2 Canonical decomposition algorithm

The *canonical decomposition* aims at generating the canonical graph representation of the proposition (Appendix C.3.1). This corresponds to the minimal set of concepts and relations needed in order to generate the complete concept graph representation of the proposition using inference and subsumption rules[1].

Parsing - This process is done recursively using a top down approach. The algorithm starts parsing the proposition in order to check its syntactical correctness. If the proposition can be parsed, the relation term and the two noun phrases are extracted, otherwise an error is returned.

Decomposing NP - The second step of the algorithm decomposes recursively both noun phrases (subsection 6.2.1). Noun phrases are decomposed first in order to remove extensions from the concept names as discussed in the subsection 6.2.2. It is important to note that the algorithm has been designed to work directly on the parse tree structures of concepts. Parse trees are reverted into string names at the last moment when creating the facts. This way, the algorithm avoids successive parsing and unparsing of concepts making the algorithm way more efficient, but also avoids the creation of some potential ambiguities (subsection 6.2.4).

Creating an observation - The algorithm creates then an *observation* using the new names returned by the decomposition. When considering a proposition of the form *NP1, R, NP2*, a fact is created as follows: *fact(prop, NP1, R, NP2, observation)* where NP1' and NP2' represents versions of NP1 and NP2 without extensions. Since passive and active voice of a proposition are not considered equivalent, the algorithm keeps the voice used. Finally, if the relation R is modified by adverbial, the PPs are removed when creating the *observation*, but a nominalization is attached to the proposition (subsection 6.2.3).

Fact notation - It can be noted that the basic structure of facts in the knowledge base has been slightly changed in order to keep track of the syntactical origin of the fact (compound noun, RC, PP, ...) and of the semantic underlying relation. Facts are now defined as follows: *fact(O, C, V, D, R)* where C and D represent concepts, V represents the explicit relation taken from a transitive verb or a preposition, O represents the syntactic origin of the word and R represents the underlying thematic relation guessed by the parser, an *observation* or a *definition.*

6.2.1 Handling NP-modifiers

An important part of the algorithm concern then the recursive decomposition of both noun phrases NP1 and NP2. The algorithm especially distinguishes the cases of noun phrases possessing exactly one modifier and of noun phrases possessing more than

[1]For instance, on the previous example, the relation *fact(definition, cell that produce insulin, produce, hormone)* is not considered as a part of the canonical graph since it is actually inferred from the relations *fact(definition, cell that produce insulin, produce, insulin)* and *isa(insulin, hormone).*

one. Noun phrases without modifiers refer to simple concepts, representing leaf nodes of the concept graph, which cannot be decomposed.

6.2.1.1 Noun phrases with exactly one modifier

Similarly to the discussion made on RC representation (chapter 3) and based on the discussions made on each of the modifiers (chapter 4), it is possible to decompose concepts with one modifier into a class inclusion relation and another relation. The table 6.1 presents a summary of these relations. In this table, the notation *[rel]* stands for the thematic relation extracted during the ontological analysis (chapter 5). The notation *[adj entity]* represents an entity having the properties described by the adjective, for instance a *red entity* is a class containing all entities of the ontology which hold the property of being red.

Modifier	**Syntax**	**Inclusion relation**	**Other relation**
RC	C-that-R-D	isa(C-that-R-D, C)	fact(prop, C-that-R-D, R, D, def)
PP	C-P-D	isa(C-P-D, C)	fact(prep, C-P-D, PREP, D, [rel])
Gerund	C's D	isa(C's D, D)	fact(ger, C's D, none, D, [rel])
Comp. Noun	C-D	isa(C-D, D)	fact(cn, C-D, none, D, [rel])
Adj - sub.	adj C	isa(adj C, C)	isa(adj C, [adj entity])
Adj - inter.	adj C	x	isa(adj C, [adj entity])
Adj - others	adj C	x	x

Table 6.1: Handling of concept modifiers.

6.2.1.2 Noun phrases with more than one modifier

The case of noun phrases with many modifiers is more complex to handle and the approach chosen towards them is what defines the decomposition as being canonical and not complete. In fact, instead of trying to generate directly all possible relations, the approach focuses on finding the main relations, the ones that belong to the canonical graph. This is achieved by generating all the concepts directly including the concept that is currently processed. Indeed, a concept represented by a noun phrase with N modifiers *Mi* is included inside each concept made of a combination of some of these modifiers. Especially, this concept is directly included in the N distinct concepts containing exactly (N-1) modifiers *Mi*. The figure 6.2 shows an example with N = 3 applied recursively.

Focusing only on the direct class inclusion enables to ease the flow of the algorithm and enables to avoid dealing with too many duplicates of thr extracted relations.

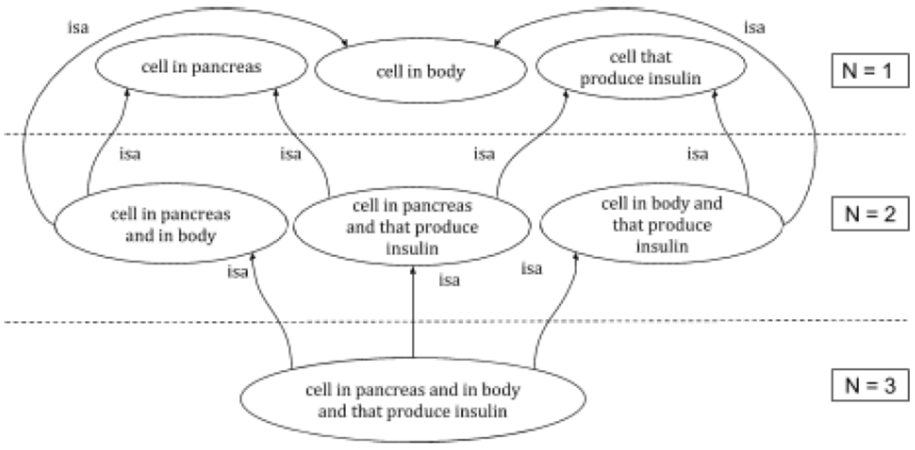

isa
cell in pancreas
cell in body
cell that produce insulin
N = 1
isa
isa
isa
isa
isa
cell in pancreas and in body
cell in pancreas and that produce insulin
cell in body and that produce insulin
N = 2
isa
isa
isa
isa
cell in pancreas and in body and that produce insulin
N = 3

Figure 6.2: Decomposition of NP with many modifiers.

6.2.2 Handling NP-extensions

An important step in the decomposition of noun phrases is the handling of extensions such as appositions and parenthetical clauses. As extensions, they show a different behaviours than concept modifiers: they add additional information on concepts without modifying them.

If extensions were handled naively as modifiers, a sentence like *cell that produce insulin, which isa hormone, regulate body* would be transformed into 6.3 instead of 6.4.

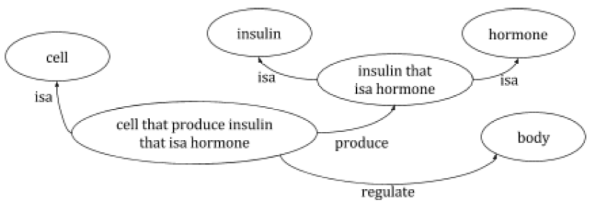


Figure 6.3: Error for representation of an apposition.

The concept graph 6.3 contains many differences with the initial expectation. In fact, when applying a rule for knowledge extraction similar to the one defined for relative clauses, the concept *insulin which isa hormone* would be decomposed in two relations *isa(insulin that isa hormone, hormone)*, and *isa(insulin,hormone)*. One additional node and one additional relation are then created. However, they do not contain any additional information, but only create noise in the graph. This noise introduces errors, because the concept nodes *insulin* and *insulin which isa hormone* appears as distinct while they actually refer to the same concept. Thus, the concepts of *insulin*

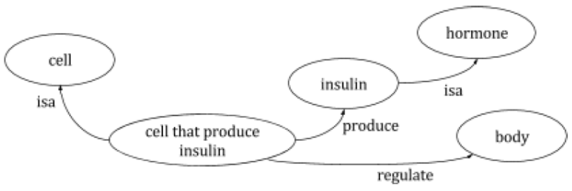


Figure 6.4: Representation of proposition with an apposition.

and *hormone* are represented as overlapping and not as a class inclusion. It can also be seen that this problem is back propagated to all concepts containing an extension in their parse tree, whatever its depth in the parse tree is. For instance, the concept node *cell that produce insulin which isa hormone* has been created instead of the concept node *cell that produce insulin.*

A specific treatment had then to be defined in order to handle concept extensions in noun phrases. This specific case is in charge of creating the correct class inclusion relation, but also to remove the extension from the parse tree, so that the back propagation problem can be avoided. As a consequence the algorithm has been designed in a post-order traversal manner so that the new names without extensions can be used when creating the facts. Extensions are then removed, starting from the leaf nodes, and corrected names are back propagated until the root.

6.2.3 Handling adverbial PPs

The handling of adverbial PPs is using a nominalization of the sentence.

Nominalization - The main step of the treatment of adverbials is then to nominalize the sentence. This can be done easily by modifying the tree structure. The subject is attached as a PP-by with an *agent* underlying relation, while a short ontological analysis is performed in order to attach the object as a PP-of with the underlying relation being either *patient* or *result.* From the proposition *betacell produce in pancreas insulin*, the tree structure for the concept *production of insulin by betacell in pancreas* can be constructed.

```
np(n(production),
   mod([pp(of, [result], np(n(insulin), mod([]), ext([]))),
        pp(by, [agent], np(n(betacell), mod([]), ext([]))),
        pp(in, [location], np(n(pancreas), mod([]), ext([])))]),
   ext([]))
```

Working over the tree structure and not over the concept name was especially important since it avoids creating referential ambiguities. In fact, in the previous example, the PP-in is linked to *production*, while it could have also been understood as linked to the *betacell* from the concept name. The nominalized tree structure is then recursively decomposed as a NP into a list of facts. Finally, the tree is unparsed in order to attach the concept name representing the nominalization to the proposition.

Temporary solution for conditions - Since conditions are not handled for now, adverbial PPs are kept in the observation created and when using the predicate attach in order to keep track of conditions. This solution is then slightly varying from the concept graph representation proposed in section 4, but avoids the rendering of logically false sentences. For instance, without adverbial PPs, *fact(prop, [betacell], [stop], [insulin secretion], observation)* would be generated from the sentence *betacell stop at low glucose concentration insulin secretion into general circulation.*

6.2.4 Reversing tree structures to string names

While tree structures represent an unambiguous concept representation, it is known that string names for concepts can be ambiguous.

Even if the implemented approach successfully avoids to parse and unparse noun phrases, a poor unparsing can create errors in the graph by bringing under the same names concept whose parse tree is actually different. This is especially the case for concepts containing many prepositional phrases like nominalizations. It is then important to check that the name returned by the unparsing is made so that successive parsing and unparsing won't affect the tree structure of the compound concept. Especially, scenarios as the following one should be avoided:

```
?− np(X,_,[liver,release,of,glucagon,in,blood,by,glycogenolysis,and,by,
   gluconeogenesis],[]), reverse_np(X,Y), np(M,_,Y,[]).

X = np(n(release),
     mod([cn([agent], n(liver)),
          pp(of,[patient], np(n(glucagon), mod([]), ext([]))),
          pp(in,[location], np(n(blood), mod([]), ext([]))),
          pp(by,[manner],np(n(glycogenolysis), mod([]), ext([]))),
          pp(by,[manner],np(n(gluconeogenesis),mod([]),ext([])))]),
     ext([])),

Y = [liver, release, of, glucagon, in, blood, by, glycogenolysis, by,
      gluconeogenesis],

M = np(n(release),
     mod([cn([agent], n(liver)),
          pp(of,[patient], np(n(glucagon), mod([]), ext([]))),
               pp(in,[location], np(n(blood), mod([]), ext([]))),
               pp(by,[manner], np(n(glycogenolysis),
                         mod([pp(by,[manner],np(n(gluconeogenesis), mod([]), ext([])))]),
               ext([]))))]),
          ext([])) .
```

A solution for this problem has been to enable the unparsing to use explicit alignment when needed. The predicate reverse_np/2 in charge of the unparsing has then been implemented in a way so that is does not only unparse, but also checks that the parsing preserves the tree structure by parsing again the created string and checking if it has changed. Based on Prolog's backtracking, a name that is stable through successive parsing is then returned. This approach has appeared to be successful on basic cases (Appendix C.5.2).

However, this also introduced a real slow down of the performances[2] and was then not been finally not been considered in the final implementation. Another solution is shortly discussed in chapter 9 for this problem of ambiguity between concepts and designation.

6.3 Graph completion based on inference rules

This section aims at defining inference rules that can be applied to Natural Logics extensions, but also to discuss shortly the implementation of an algorithm working over these rules.

6.3.1 General Inference rules

Accordingly to the discussion in Core Natural Logics, on top of the transitive isa relation, three basic inference rules are defined:

```
fact(O,Csub,V,D,R) :-isa(Csub,C), fact(O,C,V,D,R). % Inheritance.
fact(O,C,V,Dsup,R) :-isa(D,Dsup), fact(O,C,V,D,R). % Generalization.
fact(O,Csub,V,Dsup,R) :-isa(C,Csub), fact(O,C,V,D,R), isa(D,Dsup). % Both.
```

6.3.2 Inference rules for adverbial PPs

Inference rules could also be drawn based on adverbial PPs[3]. The two main cases are originating from the following example: if *betacell produce in pancreas insulin*, then *betacell produce insulin* and *betacell produce in organ insulin.* Rules can be expressed as follows:

- If (C [R-PREP-B] D) then (C R D).
- If (C [R-PREP-B] D) & (B isa B') then (C [R-PREP-B'] D).

Finally, this approach can be extended to all relations containing one or more adverbials. It is actually possible to formulate a more general inference rule, covering the two cases mentioned previously.

- For a fact F1 = (NP1, R, NP2) attached to the nominalization N1 = nomi(NP1, R, NP2), if there exists a nominalization so that N2 = nomi(NP1, R2, NP2) and isa(N1,N2), then, the fact F2 = fact(NP1, R2, NP2) can be inferred.

[2]This slow down can be explained for two reasons. First, parsing is a process that can take up to a few minutes when dealing complex propositions. Secondly, the recursive structure of the code implied to perform this parsing-unparsing potentially an important number of times.

[3]The case of adverbial PPs representing conditions is not considered here, but should be handled differently.

6.3.3 Domain dependent inference rules

Finally, domain dependent inference rules can also be defined. For instance, a relation such as *has_part* can be made transitive [2].

6.3.4 Algorithm

The first part of the implementation of the inference algorithm relies on the implementation of all the inference rules (Appendix C.3.2). After implementing them, the missing relations are generated using the built-in predicate setof/3. which returns all variable instantiations satisfying a given relation. This predicate relies on Prolog's tree structure and backtracking in order to generate the set of all possible solutions. As a set, it does not contain any duplicates. Thus, approach is then perfectly suited for a Prolog implementation.

It can also be noted that the code runs over a knowledge base, represented as a list of facts, which is given as input for each rule so that no facts need to be statically saved. This allows the algorithm to run on distinct examples without creating any potential confusions due to some saved facts.

6.4 Graph extension by Subsumption

The concept of subsumption has already been presented when dealing with Core Natural Logics. This section focuses now on extending it on the extensions considered by Extended Core Natural Logics. This is particularly appreciated since it allows to speed up the search process by extending the graphs and then creating more efficient and reliable short paths.

6.4.1 Subsumption rule

Subsumption is defined as the calculation of all logically missing relevant concepts and class inclusions. Each subsumption rule is based on one compound concept and one or many class inclusion relations taken from the knowledge base. As a consequence, a set of rules can be defined in the table 6.2. In this table, the notation C always refers to the concept being modified while D represents the concept involved in the concept modifier.

In Core Natural Logics, subsumption is based on upward monotonicity on the modifier and on the referent. Downward monotony is not considered due to the existential important assumption (subsection 3.4.3). This is actually valid for all post-modifiers (rules 1 to 6)[4], but also on pre-modifiers (rules 7 to 12)[5]. Only adjectives show a different behaviour concerning subsumption. In fact, no general rule can be defined

[4]From *betacell in pancreas* and *betacell isa cell* and *pancreas isa organ*, the concepts *cell in pancreas*, *betacell in organ* and *cell in organ* can potentially be created.

[5]From *insulin synthesis* and *insulin isa hormone* and *synthesis isa production*, the concepts *hormone synthesis*, *insulin production* and *hormone production* can potentially be created.

Rule	**Modifier**	**Comp. concept**	**Inclusion(s)**	**New concept**	**Type**
1	RC	C that R D	D isa D’	C that R D’	↑ modifier
2	RC	C that R D	C isa C’	C’ that R D	↑ concept
3	RC	C that R D	C isa C’ D isa D’	C’ that R D’	↑ both
4	PP	C in D	D isa D’	C in D’	↑ modifier
5	PP	C in D	C isa C’	C’ in D	↑ concept
6	PP	C in D	C isa C’ D isa D’	C’ in D’	↑ both
7	CN	D C	D isa D’	D’ C	↑ modifier
8	CN	D C	C isa C’	C’ D	↑ concept
9	CN	D C	C isa C’ D isa D’	C’ D’	↑ both
10	GER	D-s-C	D isa D’	C-s-D’	↑ modifier
11	GER	D-s-C	C isa C’	C’-s-D	↑ concept
12	GER	D-s-C	C isa C’ D isa D’	C’-s-D’	↑ both
13	ADJ	adj C	C isa C’	adj C’	↑ concept

Table 6.2: Subsumption rules for Extended Natural Logics..

for non-subsective and privative adjectives, while only one rule (rule 13) could be defined for subsective and intersective adjectives[6].

6.4.2 Algorithm

The algorithm aims at computing all relevant relevant missing concepts. However, this imply to go in the right order. An idea of algorithm is then to give a rank to each element. Simple concepts are of rank 0. Concepts that are related to concepts of rank 0 are considered of rank 1. Each concept is then given a rank i if the maximum rank found among the concept he is related to is i-1.

Subsumption rules are then applied by starting with concept nodes with the minimal rank. The point is to make sure that when a pair of concepts is checked for subsumption, all the concept nodes pointed to from this pair have already been processed. More details on a potential algorithm approach are available in [2].

[6]Especially, no other rules are considered since no hierarchy of adjectives is considered.

CHAPTER 7
Graph search

This section aims now at making use of the graph generated in the previous section in order to answer semantic queries.

7.1 Approach

The knowledge querying approach is based on path finding in concept graph representation. Two basic applications where the answer is provided as a pathway connecting concepts can be defined.

Searching around one concept - Firstly, this can be used in order to found key relations around one concept [2]. The figure 7.1 shows three potential starts of paths that could be followed when searching around the concept *betacell*.

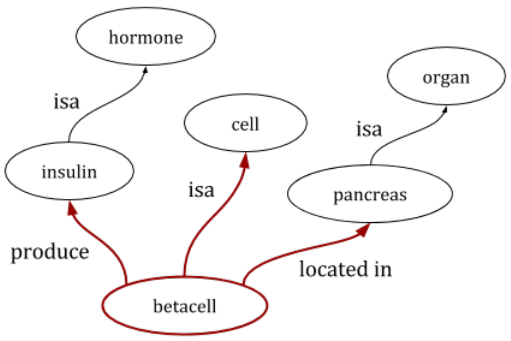


Figure 7.1: Example of knowledge base around *betacell*.

Relations between concepts - Secondly, it can also be used in order to find the relationship between two concepts [2], [3]. This pathway can be seen as an explanation of how the concepts are related. Even if this feature seems basic, it can be used when looking for definitions of the concepts and can also be used in order to check potential causal relations (like the link between *betacell* and *insulin* for instance). The consideration of only two concepts is motivated here by the refutation of the use of cumulative readings in the current version of Natural Logics.

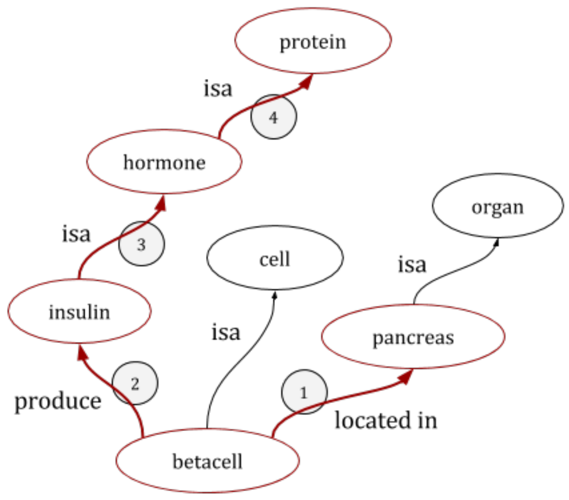


Figure 7.2: Example of knowledge base querying between *pancreas* and *protein*.

This thesis aims then also at answering queries of type Q = (C,C') such as the one presented in figure 7.2. The algorithm should then be able to find the path: *1) pancreas contain betacell 2) betacell produce insulin, 3) insulin isa hormone, 4) hormone isa protein* and should return an answer in natural language that is similar to *pancreas contain betacell which produce insulin which isa hormone which isa protein.*

7.2 Translation to Natural Language

Natural Logics relyies on a graph representation with oriented edges. However, Natural Logics does not consider the oriented graph to find relations between concepts since graph search is made in both directions. Indeed, the orientation of edges only influence the rendering of the sentences in natural language (Appendix C.4).

Direct reading - The rendering of class inclusions and of edges followed in the right orientation is straightforward [2]. The class inclusion *isa(X, Y)* is rendered as *X is a Y* and the relation *fact(T, X, R, Y)*[1] is rendered as *X R by T Y*. For instance *fact(observation, betacell, produce, insulin)* is rendered as *betacell produce by observation insulin.* If the relation isn't representing the first edge of the path, then it is transformed using an parenthetical clause involving *which*. For instance, class inclusion would be represented as *, which is a Z.* For instance, the path *1) betacell produce*

[1]Note that the fact in the prototype implementation also consider one more element to keep track of the origin of the fact as discussed in chapter 6: from a proposition (prop), from a prepositional (prep), a relative clause (rc), a compound noun (cn) or a possessive form (ger). This is mainly for debugging and tracking purposes and thus, isn't mentioned here.

insulin, 2) insulin isa hormone should be rendered as *betacell produce by observation insulin, which is a hormone.*

When considering relations issued from prepositional phrases as *fact(prep, insulin production, into, blood)*, the thematic relation inferred during the parsing is used in order to tailor the answer. As a consequence, an arc involving the preposition *into* can then be rendered in different ways depending if the preposition implies a *direction* or a *result.*

Indirect reading - When traversing an edge in the inverse direction, the rendering is different. If the edge is at the beginning of the path, then it is rendered as *some X are Y* for class inclusion or a fact as *some X are inv(R) by Y* where *inv(R)* represent the opposite voice than the one represented on the arc. If not placed at the beginning of the sentence, the rendering is made as *, whereof some are inv(R) by Y.* This can then include either passivization or activation of the verb. Since most of the relations involve active voice, inverting the relation usually represent a passivization process.

Attach predicate - Finally, another handling should be tailored in order to traverse the edges related to the predicate *attach.* A specific behaviour should be detailed for them since they involve nominalization. When considering *if C produce in B D*, should return *B hosts production of B by C.* However this hasn't been considered for now.

Path reduction - After the path is obtained, it can be also interesting not to show all the transitivity rules. Especially, inheritance can be removed during path rendering. Part of this is handled through the use of inference rules. Based on the previous example from [2], the answer *pancreatic gland produce insulin, which is a hormone, which is a protein.* could be contracted if needed into *pancreatic gland produce protein.*. However, reducing paths to shorter natural language rendering is obviously made at cost of details which might in some cases be interesting. Implementing a feature for expanding or reducing paths as wanted by the user could be useful [2].

7.3 Search algorithm

7.3.1 Formal requirements

Based on classic search theory, some requirements can be expressed. The search algorithm should:

- deal with large graph representations containing a potentially high branching factor.
- be sound and complete.
- return shortest path solution[2].
- favor the use of direct edges over indirect ones in order to return general answers.

[2]For instance, [3] discusses the use of BFS

In fact, an important difference between directed and indirect edges is their underlying structure. Direct edges have a $\forall\exists$ structure while indirect edges implying passivization or activation rely on a $\exists\exists$ structure. As a consequence, pathways containing only direct edges can be considered as giving the answer in the form of a definition while pathways involving an indirect edge only return a special case and cannot be generalized. It is the more interesting to favor the use of direct edges over indirect edges by using them only if necessary. The computation should then be considered as a graph search with weighted paths between concept nodes utilizing standard heuristic algorithms [2].

7.3.2 Implementation

However, due to time constraints, this thesis uses here IDDFS[3] since its implementation in Prolog is particularly easy, especially for the path rendering. This approach also generally handles well large graphs with large branching factors by progressively increasing the depth. Even if IDFFS algorithms can be considered as not efficient because the first steps verified many times, it should be kept in mind that the first levels of the tree are usually fast to process and their processing time becomes negligible in comparison to deeper levels. Moreover, this approach also actually helps to find a shorter solution path than a classic DFS would.

This has been used for both type of querying: around one concept and between two concepts. In both cases, it is done first on the graph considering directed edges to get the general solution first, if there exist some, then on the non oriented one in order to consider also indirect edges. No path reducing has been considered for now when processing the answer.

[3]Abbreviation of Iterative Deepening Depth First Search.

CHAPTER 8

Assessment of Natural Logics performances

Natural Logics is now supposed to be able to cope with most linguistics structures in scientific texts and to be automatically decomposed into concept graph over which graph search can be conducted.

As a consequence, this chapter focuses on assessing the performances of these different elements. It discusses the expressivity of its grammar, the accuracy of the parser, but also the performances of the graph decomposition and search algorithm[1], [2].

8.1 Correctness of Natural Logics Grammar

8.1.1 Expressiveness: Scope and missing features of the grammar

After applying the pre-processing, Natural Logics is able to capture most of the knowledge from the reference text (Appendix A). This pre-processing work is important since without it some sentences could appear as grammatically incorrect. Details on this pre-processing and on the translation of the natural language sentences into Natural Logics are available in the appendix B and D.1.1

Natural Logics extensions and the extended use of adverbial and nominal PP enable to capture most of the knowledge. In the example, conditions and comparisons are captured this way even if their handling with the graph search hasn't been discussed for now. However, some elements cannot be represented. Even if some are not needed like logical links between propositions, which are assumed to represent the reasoning process of the author more than the content itself, or adverbs like *especially* whose meaning is actually not affecting the underlying ∀∃ syntax, others need to be cap-

[1]For further details, it can be interesting to have a look at the unit tests made for the Prolog prototype (appendix C.5) since they show the basic behaviour expected for the main parts of the algorithm as well as their current implementation status.

[2]Finally, even if it wasn't the main interest of this project, it is important to note that the Prolog prototype seems computationally efficient. In fact, the parsing of most of the sentences is almost instantaneous, and the time for the longest ones remains decent. Results for the decomposition are similar. Parsing and decomposing all the sentences from the first two paragraphs takes less than 35 seconds using SWI-Prolog. Only the inference engine, dealing with a pretty broad knowledge base, is longer as it could be expected. Especially since no specific strategy to avoid processing sentences many times have been discussed. Graph querying usually answer in less than a minute. All together, this makes the application computationally usable.

tured. For instance, adverb oriented adverbs such as *strongly* or quantification adverbs such as *probably* are not handled and affect then the quality of the knowledge captured.

Finally, during the pre-processing, words possessing the same meaning were brought together under the same denomination. This is the case of the term *level* in *glucose level* that is changed into *concentration.* These changes are motivated by the inability of this prototype to deal with synonyms which represents, for now[3], a pretty important limitation that also affects graph representation and graph search.

8.1.2 Accurateness of the parsing

The accuracy of the parsing of Natural Logics propositions is computed based on two indicators. The first one refers to the accurateness of the attachment of concept modifiers. The second one refers to the percentage of correctly inferred underlying thematic relations.

The parse trees generated from the insulin article can be found in the appendix D.3 for the naive parser, and in the appendix D.4 for the ontology based parser.

8.1.2.1 Attachment disambiguation

The table 8.1 shows the results of both parsers on the 138 modifiers contained in the translation into Natural Logics of the two first paragraphs of the insulin. The sentences and some explanations about their IDs are given in the appendix D.1.1.

Naive parsing - The parsing without ontological analysis always favoring right recursion is quite accurate for sentences with a few modifiers, but fails when the intended structure is not nested (120). The parser also confuses passive sentences with agent and passive sentences without agent, but involving a manner adverbial PP-by (130 & 131). Especially, some sentences (220, 221, 232 & 233) contain the ambiguous case of an adverbial verb followed by a compound noun (cf. 4.7) and the parser deal poorly with them. All together, the naive parser shows an accuracy of 66.67%.

Ontology based parsing - The parser with ontological analysis shows interesting results (98,55%) and only commit 2 errors out of 138 modifier attachments. The first one is due to an ambiguous sentence that is wrongly resolved using Frazier's late closure principle. The other mistake refers to the attachment of a PP that is nested instead of aligned. Even if these two errors are not satisfying, the parser's decisions are matching with the default rules discussed in chapter 5 which is a good point.

8.1.2.2 Relations

The table 8.2 shows the accuracy of the ontology based parser when guessing the thematic relation between concept modifiers and their referents. Since no specific

[3]A solution is proposed in chapter 9.

Sentence ID	**Correct without ontology**	**Correct with ontology**	**Number of modifiers**
110	4	4	4
120	5	9	9
121	5	9	9
122	5	9	9
130	7	8	8
131	7	8	8
132	2	4	4
140	5	5	5
141	5	6	6
150	2	2	3
160	11	11	12
170	4	4	4
210	3	3	3
220	0	3	3
221	2	6	6
230	1	1	1
231	3	3	3
232	1	4	4
233	1	4	4
240	6	9	9
241	8	13	13
250	5	11	11
Total	**92**	**136**	**138**
Accuracy	**66,67%**	**98,55%**	

Table 8.1: Accuracy of modifiers attachment with and without ontology analysis.

implementation has been done for possessives and adjectives, this table focuses only on CNs and PPs (both adverbial and nominal).

Among the 138 modifiers discussed previously, 103 were CNs or PPs, and 96 underlying relations have been guessed correctly. In case of a miss-attached PP, like in the sentences 150 & 160, the prediction is considered false. The table 8.3 details the errors made by the parser.

8.1.3 Conclusion on Natural Logics Grammar

Even if not complete and implying a pretty important pre-processing work, Extended Natural Logics grammar seems to be expressive enough in order to capture most of the knowledge contained in a scientific texts. Especially, when dealing with a pretty narrowed domain of discourse as the one defined for this discussion, the parser using

Sentence ID	Correctly inferred thematic relation	Number of relations to infer
110	2	2
120	7	8
121	7	8
122	7	8
130	7	7
131	7	7
132	3	3
140	4	4
141	5	5
150	1	2
160	5	6
170	2	2
210	2	2
220	2	2
221	4	4
230	1	1
231	3	3
232	2	2
240	7	8
241	11	11
250	7	8
Total	**96**	**103**
Accuracy	**93,20%**	

Table 8.2: Accuracy of thematic relation inferences.

Occurence	Sentence ID	Inferred relation	Actual relation	Misinterpreted modifier
3 times	120, 121 &122	patient	beared_by	metabolism of carbohydrate
2 times	150 & 160	wrong attachment of the modifier		
1 time	240	patient	result	release of glucagon
1 time	250	bearer	location	blood concentration

Table 8.3: Summary of the wrong thematic relations.

ontological analysis shows a really good accuracy on modifier attachment, but also on inferring underlying thematic relations. However, some further challenging work is left in order to deal with possessives[4] or with compound nouns involving metonymies.

8.2 Knowledge extraction and graph representation

8.2.1 Implementation status

Canonical Decomposition - The canonical decomposition algorithm (Appendix C.3.1) has been the subject of extensive tests during the development and of a set of unit tests testing its basic features. This algorithm seems to succeed in showing the intended behaviour by decomposing successfully complex propositions into simple facts and class inclusions.

Inference Engine - The inference engine (Appendix C.3.2) aiming at computing logically relevant missing relations hasn't been tested by implementing test procedures, but seems also to perform well on the implemented parts. However, inference rules are for now only general (inheritance, generalization, transitive class inclusion) while they should also be able to deal over adverbial PPs and could also potentially include some domain specific inference rules (for instance, transitive *has_part* relation).

Subsumption Engine - The algorithm for subsumption (Appendix C.3.3) aiming at computing logically relevant concepts has however not been fully implemented due to a lack of time. The first part of the algorithm computing the rank of concepts has been implemented successfully though.

8.2.2 Conclusion on decomposition

Some parts of the algorithm haven't been implemented (subsumption feature and some domain specific rules), but the algorithm still succeed in decomposing a large part of the knowledge contained in the sentences from the article. The canonical decomposition generates a set of 1043 distinct relations, while the inference engine generates 2141 new relations. An overview of some of the inferred knowledge is available in the appendix D.5 where relations appear to be all relevant[5]. Even if the decomposition is then not complete, the algorithm provides a pretty important knowledge base that can be used for graph search.

8.3 Graph querying

The focus of this master thesis wasn't relying on the graph search algorithm. However, its results are satisfying.

[4]See example on Mary's music that can represent both *belonging* or *creation* relations depending on the context of the sentence.

[5]By going through the list of inferred knowledge, none of them seemed to be irrelevant and to be the result of miss-computations. However, tests haven't formally been performed on this part of the code.

The implemented IDFFS for graph querying succeed in finding paths between two concepts and around one central concept. The algorithm is able to render answers into natural language. Even if this rendering can be improved[6], it succeeds already in tailoring different unambiguous sentences based on the type of thematic relation between concepts, but also handles passivization when needed.

Moreover, the algorithm also tend to favor answers based on direct edges, before considering answers involving indirect ones. This is important since indirect edges result in passivization processes. Even if answers with passivization remain logically true, they only discuss some particular cases (cf. Example 3 and 4) and not general answer. Their underlying syntax is then ∃∃ and not ∀∃.

Some examples of search can be seen below.

```
% First example: basic rendering of a compound noun
?− read_from_text("insulin_kb.txt",KB), !, search(KB,5,[insulin,concentration],[insulin],R).
R = [every, [insulin, concentration], is, related, by, [bearer], to, [insulin]]

% Second example: rendering of a condition
?− read_from_text("insulin_kb.txt",KB), !, search(KB,5,[betacell],[protein],R).
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood], by, observation,
     [insulin], which, [regulate, by, promotion, of, absorption, of, glucose, from, blood, into, fat,
     into, liver, into, skeletal, muscle], by, observation, [metabolism, of, protein], which, involves,
     [protein]]

% Third example: rendering using passivization
?− read_from_text("insulin_kb.txt",KB), !,
    search(KB,5,[high,glucose,concentration],[glycogenesis],R).
R = [some, [high, glucose, concentration], is, related, by, [bearer], to, [glucose], which, is,
     affected, in, [conversion, of, absorbed, glucose, by, glycogenesis], which, is, made, by,
     means, of, [glycogenesis]]

% Fourth example: rendering using passivization
?− read_from_text("insulin_kb.txt",KB), !, search(KB,5,[high,glucose,concentration],[insulin],R).
R = [some, [high, glucose, concentration], is, a, [high, concentration], whereof, some, are, a,
     [high, insulin, concentration], which, is, related, by, [bearer], to, [insulin]]

% Fith example: request around the concept of "insulin"
?− read_from_text("insulin_kb.txt",KB), !, search(KB,5,[insulin],R).
R = [every, [insulin], is, a, [peptide, hormone, that, is, produced, by, betacell, of, pancreatic, islet]] ;
R = [every, [insulin], is, a, [hormone, that, is, produced, by, betacell, of, pancreatic, islet]]
...
```

8.3.1 Limitations

As seen in the examples above, even if all answers are logically true, some answers are not completely satisfying.

[6]For instance, the rendering of past participle in passivization.

Uninformed Graph Search - In fact, some answers are not completely satisfying due the use of the IDFFS which does not always return the best path between concepts. The best answer when querying how two concepts are related is sometimes not the first one given by the algorithm. The missing subsumption algorithm might also play a role here. Moreover, the algorithm tends to answer with sentences that are not always the most interesting ones (cf. Fourth example). Implementing the algorithm with different costs based on the type of relation and its direction could be an interesting feature for Natural Logics.

Concept denotation - The algorithm fails at recognizing terms that refer to the same concept as identical. For instance, concepts using implicit and explicit alignment or switching modifiers should be recognized as identical when they refer to the same concept. For instance, *production of glucose by liver*, *production of glucose and by liver*, *production by liver and of glucose* are all representing the same concept of production, but would be considered as distinct nodes in the graph representation. Some efforts have been made in the algorithm in charge of reversing parse trees into strings (Appendix C.2.4) in order to bring them under a same name notation that would be invariant to successive parsing and un-parsing. However, if the implemented approach was succeeding on basic cases (cf. C.5.2), it was introducing a real slow down in the performances of the algorithm[7] and has then not been considered in the final version. A solution for this problem would then be to consider the tree structure instead of the string representation of the concept when creating the graphs. This way concepts with different name, but with similar parse trees would be brought under the same node. The string name would then only be used as a user friendly parameter while the tree would actually be the concept signature.

Unimplemented features - Finally, graph querying also shows some limitations due to the lack of some features that have not been implemented yet. For instance, conditions are not considered in the graph search while they were part of the extensions of Natural Logics. A dedicated approach should be defined for them. Similarly, an approach to deal with adverbial PPs and the predicate *attach* should be defined. Especially, this approach should be able to keep somehow track of the voice used in the proposition *passive* or *active* while the nominalized form does not show it explicitly. Graph search should also be extended in order to deal with conditions. The sentence from the example 2 contains a condition and shows why the rendering of adverbial PPs was necessary and kept during the decomposition. In fact, without adverbial PPs, the answer would have been simplified, leading to a property that does not hold. An interesting feature might have been to render adverbial PPs only in the case of conditions. Later on, graph querying should also be extended in order to deal with the new features for Natural Logics such as comparisons or temporal reasoning.

[7]By introducing the parsing of noun phrases as a verification after its un-parsing

8.3.2 Conclusion on graph search

The graph querying algorithm is then satisfying as a first prototype implementation. Even if some major limitations have been highlighted, they are not insurmountable and some potential solutions have already been discussed. Further work would then be needed, especially in order to deal with adverbial PPs and their attached nominalization.

CHAPTER 9

Conclusion

This chapter aims now at summarizing the work done during this thesis, discussing the results obtained and the potential improvement that are left as future work.

9.1 Summary

This report started by analyzing knowledge representation in natural language based on ontology and syllogistic logic. This helped to define a simple approach for the underlying syntactic structure of natural language sentences. This was the basis of the definition of Core Natural Logics which allows the representation of simple concept, but also nested and aligned compound ones. After defining a set of extensions on modifiers and on relations, a grammar and a parser able to cope with the structure of most sentences of scientific texts have been defined. However, the increasing complexity generated by these extension is also the source of various ambiguities. As a consequence, an approach for syntactic disambiguation has been defined. It relies on an ontological analysis to define constraints for verbs and adjectives and concept affinities for compound nouns, nominal PPs and adverbial PPs. An algorithm to decompose complex Natural Logics propositions into concept graph representation has then been defined. This algorithm especially includes an inference and a subsumption engine to compute logically relevant missing concepts and relations. Finally, a graph search algorithm working over concept graph representations of propositions has been defined in order to answer queries on the relations between two concepts.

This thesis proposes then a prototype able to answer questions on the relation between concepts related to human insulin production. Especially, the prototype answers using Natural Language, unlike FOL or DL, it is then also usable by inexperienced users.

9.2 Discussion

From a theoretical point of view, the approach of Natural Logics goes in the same direction that a pretty important number of researches on the semantic of the English language. By working directly with Natural Language as vessel to carry information, Natural Logics goes beyond Description Logics on some aspects. Particularly, it enables to capture the meaning of adverbials, to work over specific $\forall\exists$ and $\exists\exists$ sentence constructions and makes it usable by everyone without any prerequisite knowledge.

The work made during this thesis on the extended grammar for Natural Logics is particularly interesting since it allows a pretty broad expressivity while being almost not ambiguous inside the domain of discourse. When testing the parser, 98.55% of modifiers were properly attached and 95.0% of them[1] were associated with the right thematic relation. This means that the parsing and attachment of modifiers is accurate, but also that the reasons of modifier attachments are correctly captured by the ontological model. However, it remains still unsure that this approach could be extended successfully as such in order to deal with a larger domain of discourse. The definition of the set of constraints and affinities is here particularly important. In fact, if constraints are made too general, the parser will tend to allow semantically incorrect structures. Oppositely, if the constraints are too specific, the parser will tend to discard some unusual semantically correct structures. This approach should then be tested using a larger set of constraints and affinities to deal with a larger domain of discourse. Improving the underlying ontology at the same time then updating the constraints and affinities appear then as a challenging, but interesting task.

Graph decomposition and graph search, even if not complete in this project,are also interesting since they show it is possible to transform complex propositions into simple facts and class inclusions that can be then used for graph querying. Even if the implemented graph search isn't optimal, it works as a proof of concept for knowledge querying by showing that queries can be answered in natural language.

9.3 Future works

Without considering the potential extension of this approach to a larger domain of discourse, some work is left over in this thesis, especially in the implementation part.

Implementation of missing features - Before considering new extensions for Natural Logics, a first step in order to improve the current prototype and keep on testing Natural Logics would be the implementation of the missing features. In fact, not all extensions described have been completely discussed, such as the handling of conditions, or have not been completely implemented such as the inference rules on adverbial PPs or the subsumption algorithm.

Separating the concept from its designation - Another future area of work in Natural Logics reside in the distinction of denotation and connotation. As explained previously, Natural Logics fails to recognize some similar concepts as identical when expressed differently. For instance, *synthesis by liver of glycogen*, *synthesis of glycogen by liver* and *synthesis by liver and of glycogen* and *synthesis of glycogen and by liver* all represent the same concept, but have different denotation depending on the order in which modifiers are written and considering or not the presence of explicit alignment.

[1]Among 101 compound nouns and prepositional phrases, correctly attached, 96 were associated with the correct thematic relation.

This problem is actually similar to the inability of the system to handle polysemy and synonyms. In fact, the implementation made previously only consider a 1-1 relation between denotations and concepts themselves[2], while the relation is actually a m-n relation where m and n are both superior or equal to 1.

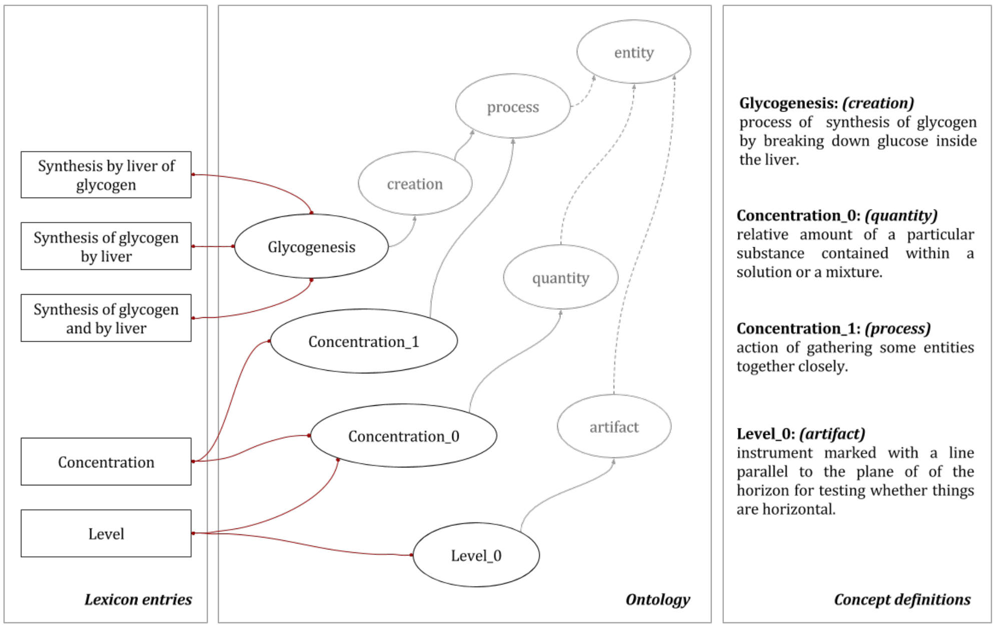

entity
process
creation
Synthesis by liver of glycogen
Synthesis of glycogen by liver
Synthesis of glycogen and by liver
Glycogenesis
quantity
Concentration_1
Concentration
Concentration_0
artifact
Level
Level_0
Lexicon entries
Ontology
Glycogenesis: (creation) process of synthesis of glycogen by breaking down glucose inside the liver.
Concentration_0: (quantity) relative amount of a particular substance contained within a solution or a mixture.
Concentration_1: (process) action of gathering some entities together closely.
Level_0: (artifact) instrument marked with a line parallel to the plane of of the horizon for testing whether things are horizontal.
Concept definitions

Figure 9.1: Intermediate layer to distinguish ontology concepts from lexicon entries.

A solution to solve this problem is to define a new predicate bringing together the denotation of a concept and the concept itself referring to a node of the ontology. This approach is explained in [25], defining a multi-valued relation from the lexicon entries to the concepts in the ontology (and inversely) that is able to handle synonyms and word polysemy. The figure 9.1 attends to explain it pictorially by showing the synonymy between *concentration* and *level*, the polysemy of *concentration* and the possibility to refer to one same concept using different notations (for *glycogenesis*).

This approach should then be extended into the concept graph representation in order to ensure the unicity of the representation of each concept in the graph.

Add further extensions to Natural Logics - After implementing the missing features and the intermediate layer, some further extensions could be added to Natural Logics in order to extend its scope. For instance, quantification and time dependencies for

[2]This is also why each word is associated to only one lexicon entry and is assigned to only one ontological type (Appendix D.1.2).

time reasoning could be long term challenging goals. The handling of metonymies seems also to be an important point in order to later on perfectly handle some Natural Logics features . It would be especially interesting for to improve the ontological analysis around compound nouns and possessives. This could also represent a first step in order to extend the scope of Natural Logics to other texts than scientific ones.

Integration of Natural Logics - Finally, some further work could also be done in order to think about Natural Logics integration in larger systems. It could be interesting to work on estimating the workload needed to implement an algorithm for sentence pre-processing. In fact, the expressivity of Natural Logics grammar appears to be closely related to the pre-processing. Including new features to the parser might then help to implement the pre-processing, making then the global system more robust. The integration of this approach in a text processing treatment also needs to be considered to define in which order sentences are processed. In fact, some features as the handling of disjunctions or the inference rules depends on the already existing knowledge base. Thus, it is possible to imagine a scenario where the common supremum of a disjunction is not present in the knowledge when the sentence is processed, but appears only later on. The parsing of the sentence fails while it should have succeed. Defining an optimal way to process sentences and to maximize the success of the parsing is then valuable.

Conclusion - Natural Logics appears then as a solution for semantic reasoning working over natural language. Even if a lot of work stays ahead concerning both the implementation of its features related to semantic search and to its integration in other systems, the results given by the prototype implemented during this thesis are promising.

APPENDIX A
Reference text: Wikipedia's Article on Insulin

The reference text used in order to test Natural Logics approach for knowledge representation and semantic analysis is the *insulin* article available on Wikipedia [39]. Especially, this master thesis aims at capturing the knowledge related to human insulin production contained in the two first paragraphs of the article.

As a consequence, some changes have been introduced in the article: elements related to non-human insulin production, to etymology of terms or to insulin treatments have been deleted.

A.1 Original version

Insulin (from the Latin, insula meaning fish) is a peptide hormone produced by beta cells of the pancreatic islets, and by the Brockmann body in some teleost fish. It has important effects on the metabolism of carbohydrates, fats and protein by promoting the absorption of, especially, glucose from the blood into fat, liver and skeletal muscle cells. In these tissues the absorbed glucose is converted into either glycogen via glycogenesis or fats (triglycerides) via lipogenesis, or, in the case of the liver, into both. Glucose production (and excretion into the blood) by the liver is strongly inhibited by high concentrations of insulin in the blood. Circulating insulin also affects the synthesis of proteins in a wide variety of tissues. In high concentrations in the blood it is therefore an anabolic hormone, promoting the conversion of small molecules in the blood into large molecules inside the cells. Low insulin levels in the blood have the opposite effect by promoting widespread catabolism.

The pancreatic beta cells (βcells) are known to be sensitive to the glucose concentration in the blood. When the blood glucose levels are high they secrete insulin into the blood; when the levels are low they cease their secretion of this hormone into the general circulation. Their neighboring alpha cells, probably by taking their cues from the beta cells, secrete glucagon into the blood in the opposite manner: high secretion rates when the blood glucose concentrations are low, and low secretion rates when the glucose levels are high. High glucagon concentrations in the blood plasma powerfully stimulate the liver to release glucose into the blood by glycogenolysis and gluconeogenesis, thus having the opposite effect on the blood glucose level to that produced by high insulin concentrations. The secretion of insulin and glucagon into the blood in response to the blood glucose concentration is the primary mechanism responsible for keeping the glucose levels in the extracellular fluids within very narrow limits at rest, after meals, and during exercise and starvation.

A.2 Shortened version

Insulin is a peptide hormone produced by beta cells of the pancreatic islets. It regulates the metabolism of carbohydrates, fats and protein by promoting the absorption of, especially, glucose from the blood into fat, liver and skeletal muscle cells. In these tissues the absorbed glucose is converted into either glycogen via glycogenesis or fats (triglycerides) via lipogenesis, or, in the case of the liver, into both. Glucose production (and excretion into the blood) by the liver is strongly inhibited by high concentrations of insulin in the blood. Circulating insulin also affects the synthesis of proteins in a wide variety of tissues. It is therefore an anabolic hormone, promoting the conversion of small molecules in the blood into large molecules inside the cells. Low insulin levels in the blood have the opposite effect by promoting widespread catabolism.

Pancreatic beta cells (β cells) are known to be sensitive to glucose concentrations in the blood. When high, β cells secrete insulin into the blood; when low they cease their secretion. Their neighboring alpha cells, by taking their cues from the beta cells, secrete glucagon into the blood in the opposite manner: increased secretion when blood glucose is low, and decreased secretion when glucose levels are high. Glucagon, through stimulation the liver to release glucose by glycogenolysis and gluconeogenesis, has the opposite effect of insulin. The secretion of insulin and glucagon into the blood in response to the blood glucose concentration is the primary mechanism responsible for keeping the glucose levels in the extracellular fluids.

APPENDIX B
Theoretical Scope of Extended Natural Logics

The objective of this section is to analyze the scope of the linguistic structures that can be captured by Natural Logics in scientific texts based on the two first paragraphs of the Wikipedia article on *Insulin* [39] (appendix A).

B.1 Pre-processing assumptions

It is assumed here that Extended Natural Logics handles successfully the extensions defined in chapter 4.

Before translating the sentences into Natural Logics, a pre-processing is applied in order to rephrase sentences so that they would be handled in a better way by Natural Logics. Among others:

- **Pronoun disambiguation:** In order to avoid dealing with reference ambiguities which are known as a very challenging area in computational linguistics, pronouns are explicitly replaced by their reference. In case of demonstrative pronouns referring to the previous sentence, a nominalization of the sentence is used.

- **Lexicon simplification:** In order to avoid dealing with synonyms and the polysemy of words, the lexicon of the article is narrowed. For instance, the word *level* is replaced by the word *concentration* which is also used in the article. This pre-processing aims then at unifying the sentences around a common vocabulary.

- **Removing idiomatic expressions:** Idiomatic expressions are expressions which have a figurative meaning differing from their literal one. They are usually more frequently used by native speaker and are harder to deal with than non-idiomatic expressions. As a consequence, some of them are removed and substituted by a simpler equivalent. For instance *known to be sensitive* is replaced by *is affected by*.

- **Conjunctions & disjunctions:** Extended Natural Logics is assumed to successfully handle disjunctions and conjunctions on the subject and object of sentences. However, it cannot handle disjunctions placed inside prepositional phrases or relative clauses. Similarly, conjunctions and disjunctions of more than two elements cannot be handled. As a consequence, they are manually distributed over many similar sentences in the pre-processing.

- **Conditions:** Accordingly to Natural Logics extensions discussion, conditions are transformed into adverbial PPs during the pre-processing. For instance, the sentence: "When glucose level is high, *β-cell secrete insulin* is turned into *β-cell secrete at high glucose level insulin.*
- Finally, elements that cannot be handled by Natural Logics are not represented. This notably includes logical links between sentences such as *therefore* or *also.*

B.2 Translation into Natural Logics

B.2.1 First paragraph

The set of sentences for the first paragraph is:

1.0 Insulin is a peptide hormone produced by beta cells of the pancreatic islets.

2.0 It has important effects on the metabolism of carbohydrates, fats and protein by promoting the absorption of, especially, glucose from the blood into fat, liver and skeletal muscle cells.

3.0 In these tissues the absorbed glucose is converted into either glycogen via glycogenesis or fats (triglycerides) via lipogenesis, or, in the case of the liver, into both.

4.0 Glucose production (and excretion into the blood) by the liver is strongly inhibited by high concentrations of insulin in the blood.

5.0 Circulating insulin also affects the synthesis of proteins in a wide variety of tissues.

6.0 It is therefore an anabolic hormone, promoting the conversion of small molecules in the blood into large molecules inside the cells.

7.0 Low insulin levels in the blood have the opposite effect by promoting widespread catabolism.

After applying the pre-processing assumptions, the following set of sentences represent the translation into Natural Logics[1]:

1.0 (Insulin) isa (peptide hormone that is produced by betacell of pancreatic islet).

2.0 (Insulin) regulate by promotion of absorption of glucose from blood into fat, into liver and into skeletal muscle (metabolism of carbohydrate).

2.1 (Insulin) regulate by promoting absorption of glucose from blood into fat, into liver and into skeletal muscle (metabolism of fat).

[1]Note that brackets around the subject and the object have been added to ease the reading (they are not needed in Natural Logics).

2.2 (Insulin) regulate by promoting absorption of glucose from blood into fat, into liver and into skeletal muscle (metabolism of protein).

3.0 (Absorbed glucose) is converted in liver, in fat and in skeletal muscle cell into glycogen via glycogenesis ( ).

3.1 (Absorbed glucose) is converted in liver, in fat and in skeletal muscle into fats, which isa triglyceride, via lipogenesis ( ).

3.2 (Absorbed glucose) is converted in liver into glycogen and into fat ( ).

4.0 (Glucose production by liver) is strongly inhibited by (high insulin concentration in blood).

4.1 (Glucose excretion into blood by liver) is strongly inhibited by (high insulin concentration in blood).

5.0 (Circulating insulin) affect (synthesis in a variety of tissues of protein).

6.0 (Insulin) isa at high insulin concentration in blood (anabolic hormone that promote conversion of small molecule in blood into large molecule in cell.

7.0 (Insulin) promote at low insulin concentration (widespread catabolism).

Scope covered - After pre-processing, most of the knowledge of the paragraph has been captured by Natural Logics. The sentences contain a lot of adverbial and nominal PP which can be handled by Natural Logics. The expression of the sentences is changed, but most of the knowledge is captured without loss. For instance, some adverbs have been deleted on the way. However, it includes *also* and the adverb *especially* in the sentence *the absorption of, especially, glucose* which informs that glucose is not exclusively absorbed. However,based on the closed world assumption, if other elements are absorbed, they would be represented in the knowledge base as well. This element is then not needed. The subject oriented adverb *strongly* is not handled though.

B.2.2 Second paragraph

The set of sentences for the second paragraph is:

1.0 Pancreatic beta cells (β-cells) are known to be sensitive to glucose concentrations in the blood.

2.0 When blood glucose levels are high, β-cells secrete insulin into the blood; when the blood glucose levels are low they cease their secretion into the general circulation.

3.0 Their neighboring alpha cells, probably by taking their cues from the beta cells, secrete glucagon into the blood in the opposite manner: high secretion when blood glucose is low, and decreased secretion when glucose levels are high.

4.0 High glucagon concentrations in the blood plasma powerfully stimulate the liver to release glucose into the blood by glycogenolysis and gluconeogenesis, thus having the opposite effect on the blood glucose level to that produced by high insulin concentrations.

5.0 The secretion of insulin and glucagon into the blood in response to the blood glucose concentration is the primary mechanism responsible for keeping the glucose levels in the extracellular fluids within very narrow limits at rest, after meals, and during exercise and starvation.

The following set of sentences represent the translation into Natural Logics.

1.0 (Pancreatic betacell) is affected by (glucose concentration in blood).

2.0 (Betacell) secrete at high glucose concentration into the blood (insulin).

2.1 (Betacell) stop at low glucose concentration (insulin secretion into general circulation).

3.0 (Alphacell, neighbours of beta cell,) take cures from betacells.

3.1 (Alphacell) secrete into blood and in opposite manner as beta cell (glucagon).

3.2 (Alphacell) secrete highly at high glucose concentration (glucagon).

3.3 (Alphacell) secrete lowly at low glucose concentration (glucagon).

4.0 (High glucagon concentration in blood plasma) stimulate powerfully (liver release of glucose in the blood by glycogenolysis and by gluconeogenesis).

4.1 (Stimulation of liver glucose release in the blood by glycogenolysis and by gluconeogenesis) affect in opposite way to effect that is produced by high insulin concentration (blood glucose concentration).

6.0 (Insulin secretion into the blood in response to blood glucose concentration) and (secretion of glucagon into blood in response to blood glucose concentration) isa (primary mechanism that (keep within narrow limits at rest and after meals and during exercise and during starvation) glucose concentration in extracellular fluid).

Scope covered - By use of adverbial prepositions to express conditions and by splitting conjunctions into many simple proposition, most of the content is captured in this paragraph as well. Even if elements like comparisons are not handled for now by Natural Logics graph representation, they have still been captured as PP. For instance, *in the opposite manner* is turned into *opposite manner as betacell.* It could be extended by a noun phrase for more explicit knowledge representation: *in opposite manner as beta cell glucagon production.* Some other elements are missing though. The quantification adverb *probably* is not represented including somehow a representation mistake. Similarly possessive adjectives are not represented.

APPENDIX C Prolog implementation

C.1 Introduction to DCG in Prolog

This appendix is inspired from the book Learn Prolog Now! and its online tutorial [9]. It provides a short introduction to Context Free Languages and Definite Clause Grammar in Prolog for inexperienced readers.

Context Free Rule - In order to represent sentences such as *every beta-cell produce some insulin*, the syntactic structure *Quantifier Concept Transitive-Verb Quantifier Concept* can be defined. This structure can be implemented in Prolog as a context free rule:

$$S \rightarrow Q, C, V, Q, C.$$

Context Free Grammar - A context free rule defines under which condition a part of a sentence is syntactic correct. The symbol → is used to define a rule, specifying that the sentence S, is syntactically correct if composed of a succession of a quantifier (Q), a concept (C), a verb (V), another quantifier (Q) and another concept (C). This context free rule defined earlier can be extended, defining then a Context Free Grammar (CFG):

$$\begin{gathered} S \rightarrow Q, C, V, Q, C. \\ C \rightarrow \text{betacell} \\ C \rightarrow \text{insulin} \\ Q \rightarrow \text{every} \\ Q \rightarrow \text{some} \\ V \rightarrow \text{produce} \end{gathered}$$

On the one hand, this basic set of rules explains how grammatical categories are built up and what they are composed of. It can then be used in order to define if a sentence is grammatically correct, and if it is, why. This is done by a process of parsing. Based on this CFG, the sentence *every betacell produce some insulin* appears to be grammatical and can be turned into the following parse tree:

S (Q (every) C (betacell) V (produce) Q (some) C (insulin))

On the other hand, such rules can also be used in order to generate all grammatically correct sentences. This set of all sentences generated by the grammar, is called the language generated by the grammar. In this case, 16 sentences like *every betacell* *produce some insulin* or *some betacell produce every insulin* can be produced.

Context Free Language - A Context Free Language is a language which can be generated by a context free grammar. In this thesis, English is assumed to be a context free language.

Context Free Parser - A context free recognizer is a program which tells s whether or not a string belongs to the language generated by the grammar. A context free parser is a program which does the same but also explains what the structure of the string is. Thus, we will be particularly interested in context free parsers. In Prolog, a pretty naive way to represent context free grammar is done by representing sentences as lists and by defining if-rules using the predicate append/3. By introducing the notation NP (for noun phrase) and VP (for verbal phrase), the context free grammar can be implemented as follows:

```prolog
s(Z) :-np(X), vp(Y), append(X,Y,Z).
vp(Z) :-v(X), np(Y), append(X,Y,Z).
np(Z) :-q(X), c(Y), append(X,Y,Z).

c([betacell]).
c([insulin]).
q([every]).
q([some]).
v([produce]).
```

Difference lists - Despite a pretty good readability and understandability, this representation is actually not efficient computationally due to the use of the predicate append/3. In order to improve this, difference lists are used. Instead of representing a sentence by only one list, each sentence is represented by two. The first one contains what should be consumed, the second one, what should be left behind.

Even if this notation seems to be more complicated, it actually allows us to write our context free grammar without using the predicate append. Especially, Prolog includes a notation dedicated for difference lists in Definite Clause Grammar. After introducing the concept of lexicon to split the rules from the lexicon itself, the grammar can then be simplified as follows:

```prolog
s --> np, vp.
vp --> v, np.
np --> q, c.
c --> [C], {lex(C,concept)}.
q --> [Q], {lex(Q,quantifier)}.
v --> [V], {lex(V,verb)}.

% Lexicon.
lex(cell, concept).
lex(betacell, concept).
lex(insulin, concept).
lex(produce, relation).
```

These principles, even if quite basic, will actually be important and re-used through all the thesis.

C.2 Disambiguated Extended Natural Logics Grammar

C.2.1 Extended Grammar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% grammar.pl:
%
% Disambiguated version of Extended grammar for Natural Logics,
% based on onthological analysis.
%
%
%
% This file defines a DCG parser for Natural Logics propositions.
% Among the extensions discussed for Natural Logics, it includes:
% − fully recursive concept definition
% −  concept−modifiers:
% * relative clauses (RC)
% * prepositional phrases (PP)
% * compound nouns (CN)
% * possessives (GER)
% * adjectives (ADJ)
% −  concept extensions:
% * appositions (APP)
% * parenthetical clauses (PC)
% −  relation modifiers:
% * active and passive forms
% * adverbials PPs
% * manner oriented adverbs
% * conditions using adverbial PPs
% −  concept disjunctions
% −  concept conjunctions
%
%
% However, some features have not been considered and implemented here,
% such as prepositional verbs, passive relative clauses without
% agent or disjunctions on both subject and object of proposition.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-[ontology_skeleton]. % Loading the ontology skeleton.
:-[ontology_constraints]. % Loading the domain constraints.
:-[lexicon_insulin]. % Loading the dedicated lexicon.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DCG Parser for natural logics.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Propositions with plural formation.
p([p(NP,VP)]) −−> np(NP, const(S)), vp(VP, const(S)), {VP\=[_,_]}.
```

```
p([p(N1,VP),p(N2,VP)]) --> np(N1, const(S)), [and], np(N2, const(S)),
                                   vp(VP, const(S)), {VP\=[_,_]}.
p([p(NP,V1),p(NP,V2)]) --> np(NP, const(S)), vp(VP, const(S)), {VP=[V1,V2]}.
p([p(N1,V1),p(N1,V2),
   p(N2,V1),p(N2,V2)]) --> np(N1, const(S)), [and], np(N2, const(S)),
                                   vp(VP, const(S)), {VP=[V1,V2]}.


vp(vp(V,pred(A)), const(S0)) --> rterm(V,_,copular, const(_,_)), adjs(A, const(S1)),
                                       {isa_of(S0,S1)}.
vp(vp(V,N), const(S0)) --> rterm(V,trans,_, const(S1,S2)), np(N, const(S3)),
                                       {isa_of(S0,S1), isa_of(S3,S2)}.
vp(vp(verb(passive(V),mod(M))), const(S0)) --> [is,V],
        {lex(V0,trans, _,V,const(_,SS0))}, {isa_of(S0,SS0)}, pps(M, V0).
vp([vp(V,N1),vp(V,N2)], const(S0)) --> rterm(V,trans,_, const(S1,S2)),
        np(N1, const(S3)), [and], np(N2, const(S4)),
        {isa_of(S0,S1), isa_of(S3,S2), isa_of(S4,S2)}.


% Adjectives as predicate.
adjs([A], const(S0)) --> adj(A,predi, const(S0)).
adjs([A|T], const(S0)) --> adj(A,predi, const(S0)), adjs(T,const(S0)).


% Concept nouns accepting modifiers.
np(np(N,mod([]),ext([])), const(S0)) --> n(N, const(S0)).
np(np(N,mod(M),ext(E)), const(S0)) --> pre(Q, L, const(S0)), n(N, const(S0)),
                                             post(P,E, const(S0)),
                                             {check_adj(S0,L), append(Q,P,M)}.

check_adj(_,[]) :-true.
check_adj(X,[H]) :-isa_of(X,H).
check_adj(X,[H|T]) :-isa_of(X,H), check_adj(X,T).

% Defining pre-modifiers.
pre(T, L, const(S0)) --> pre1(T, L, const(S0)). % Case without possessive.
pre([G|T],L, const(S0)) --> cn(N1, const(_)), ["s"], pre1(T,L, const(S0)),
                                         {G = ger([affiliation],N1)}.
pre1(T, [],const(S0)) --> pre2(T, const(S0)).
pre1([adj(A)|T],[S1|L],const(S0)) --> adj(A,_,const(S1)), pre1(T,L,const(S0)).
pre2([], const(_)) --> {true}. % Potentially, no pre-modifiers.
pre2([cn([R],N1)|T], const(S0)) --> n(N1, const(S1)), pre2(T, const(S0)),
                                         {aff(cn,SS1,SS0,R), isa(S0,SS0), isa(S1,SS1)}.


% To allow compound nouns in possessives.
cn(np(N,mod([]),ext([])), const(S0)) --> n(N, const(S0)).
cn(np(N,mod(M),ext([])), const(S0)) --> pre1(M, L, const(S0)), n(N, const(S0)),
        {check_adj(S0,L)}.


% Defining post-modifiers.
post(M,[], const(S0)) --> post1(M, const(S0)). % Case without apposition.
post(M,[A], const(S0)) --> app(A, const(S0)), post1(M, const(S0)).

post1([], const(_)) --> {true}. % Potentially, no post-modifiers.

```

```
post1([rc(V,NP)], const(S0)) --> [that], rterm(V,trans,_, const(L0,L1)),
                                     np(NP, const(S1)),
                                     {reverse_rterm(V,Vs), Vs \= [isa]},
                                     {isa_of(S0,L0), isa_of(S1,L1)}.
post1([pp(P,[R],NP)], const(S0)) --> prep(P), np(NP, const(S1)),
                                     {aff(P,SS0,SS1,R), isa(S0,SS0), isa(S1,SS1)}.
post1([rc(V,NP)|M], const(S0)) --> [that], rterm(V,trans,_, const(L0,L1)),
                                     np(NP, const(S1)), align(_), post1(M, const(S0)),
                                     {reverse_rterm(V,Vs), Vs \= [isa],
                                     isa_of(S0,L0), isa_of(S1,L1)}.
post1([rc(V,NP)|M], const(S0)) --> [that], rterm(V,trans,_, const(L0,L1)),
                                     np(NP, const(S1)), post1(M, const(S0)),
                                     {reverse_rterm(V,Vs), Vs \= [isa],
                                     isa_of(S0,L0), isa_of(S1,L1)}.
post1([pp(P,[R],NP)|M], const(S0)) --> prep(P), np(NP,const(S1)), align(_),
                                     post1(M,const(S0)),
                                     {aff(P,SS0,SS1,R), isa(S0,SS0), isa(S1,SS1)}.
post1([pp(P,[R],NP)|M], const(S0)) --> prep(P), np(NP, const(S1)), post1(M, const(S0)),
                                     {aff(P,SS0,SS1,R), isa(S0,SS0), isa(S1,SS1)}.


% Defining appositions.
app(ap(N), const(_)) --> [","], [a], np(N, const(_)), [","].
app(ap(N), const(_)) --> [","], [an], np(N, const(_)), [","].


% Defining parenthetical clauses.
app(pc(V,N), const(S0)) --> [","], [which], rterm(V,trans,_,const(L0,L1)),
        np(N,const(S1)), [","], {isa_of(S0,L0), isa_of(S1,L1)}.


% Relation terms
rterm(verb(V,mod(M)),X,T, const(S,C)) --> rterm1(V,X,T, const(S,C)),
              advs(P), pps(Q,V), {append(P,Q,M)}.
rterm(verb(passive(V),mod([])),X,T, const(S,C)) --> [is,V],
              {lex(_,X,T,V,const(C,S))}, [by].
rterm(verb(passive(V),mod(M)),X,T, const(S,C)) --> [is,V],
              {lex(V0,X,T,V,const(C,S))}, pps(M,V0), [by].
rterm1(active(V),X,T,const(S,C)) --> [V], {lex(V, X, T, _, const(S,C))}.
% Space for prepositional verb here.


% Adverbial PPs.
pps([],_) --> {true}. % Potentially no adverbial PPs.
pps([pp(P,[R],N)],V0) --> prep(P), np(N, const(S1)),
        {correct(V0,V), nomi(N0,V), lex(N0,noun,S0),
         aff(P,SS0,SS1,R), isa(S0,SS0), isa(S1,SS1)}.
pps([pp(P,[R],N)|T],V0) --> prep(P), np(N, const(S1)), align(_), pps(T, V0),
        {correct(V0,V), nomi(N0,V), lex(N0,noun,S0),
         aff(P,SS0,SS1,R), isa(S0,SS0),isa(S1,SS1)}.
pps([pp(P,[R],N)|T],V0) --> prep(P), np(N, const(S1)), pps(T, V0),
        {correct(V0,V), nomi(N0,V), lex(N0,noun,S0),
         aff(P,SS0,SS1,R), isa(S0,SS0), isa(S1,SS1)}.

correct(active(V),V) :-!.
correct(passive(V),V) :-!.
correct(V,V).
```

```

% Adverbs.
advs([])  --> {true}. % Potentially no adverbs.
advs([adv(A)|T]) --> adv(A), advs(T).

% Alignment keywords.
align(X) --> [X], {X = and}.
align(X) --> [X], {X = ","}.

% Access to lexicon entries
prep(P)  --> [P], {lex(P, preposition)}.
n(n(N), const(S0)) --> [N], {lex(N, noun, S0)}.
adj(A,T, const(L)) --> [A], {lex(A, adj, _, T, const(L))}.
adv(A)  --> [A], {lex(A, adv)}.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining transitive rules for constraints.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% isa(X,Y) - returns true if X is a subclass of Y.
isa(X,X).
isa(X,Y) :-isa(X,Y,[]).
isa(X,Y,V) :-is_subset_of(X,Y), \+ memberchk((X,Y),V).
isa(X,Y,V) :-is_subset_of(X,Z), \+ memberchk((X,Z),V), isa(Z,Y,[(X,Y)|V]).

% isa_of(X,L) - returns true if X is a subclass of one of the elements of L.
isa_of(_,[]) :-fail.
isa_of(X,[H|T]) :-(isa(X,H) -> true ; isa_of(X,T)).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Helper functions
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

supremum(S1,S2,N) :-(isa(S1,N), isa(S2,N) -> print_common(N,S1,S2) ; fail).

print_common(N,S1,S2) :-nl, write("Common supremum found for "), write(S1),
                        write(" and "), write(S2), write(" with "), write(N).

reverse_rterm(verb(active(V),mod(_)), [V]).
reverse_rterm(verb(passive(V),mod(_)), [V]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.2.2 Ontology Skeleton

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ontology_skeleton.pl:
%
% Onthology sklеleton for onthological
% disambiguation of Natural Logics Grammar.
%
%
% This file aims at defining a top ontology for concepts related to
% human insulin production based on BFO. Domain specific concepts
% are then created based on the insulin article considered as a
% reference. Some relations in the file below have been commented in
% order to simplify the model and thus, to increase the efficiency of
% the parsing.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Ontology skeleton inspired from Basic Formal Ontology (BFO).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

is_subset_of(continuant, entity).
is_subset_of(occurent, entity).

is_subset_of(independent_continuant, continuant).
is_subset_of(material, independent_continuant).
is_subset_of(immaterial, independent_continuant).

is_subset_of(gen_dependent_continuant, continuant).
is_subset_of(information, gen_dependent_continuant).
% is_subset_of(gene_sequence, gen_dependent_continuant).

is_subset_of(spe_dependent_continuant, continuant).
is_subset_of(quality, spe_dependent_continuant).
is_subset_of(quantity,quality).

% is_subset_of(function, spe_dependent_continuant).
% is_subset_of(disposition, function).
% is_subset_of(regulation, disposition).

is_subset_of(process, occurent).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Domain specific ontology for human insulin production.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% is_subset_of(immaterial, independent_continuant).
% is_subset_of(cognitive, immaterial).
% is_subset_of(idea, cognitive).
```

```

is_subset_of(animate, material).
% is_subset_of(person, animate).
% is_subset_of(doctor, person).

is_subset_of(inanimate, material).
is_subset_of(countable, inanimate).
is_subset_of(mass, inanimate).
% is_subset_of(artifact, countable).
is_subset_of(body_part, countable).
is_subset_of(organ, body_part).
is_subset_of(cell, body_part).
is_subset_of(body_fluid, mass).
is_subset_of(body_secretion, body_fluid).
is_subset_of(protein, body_secretion).

is_subset_of(state, occurent).
% is_subset_of(presence, state).
% is_subset_of(lack, state).

% is_subset_of(event, occurent).
% is_subset_of(action, event).
% is_subset_of(treatment, action).

is_subset_of(creative, process).
is_subset_of(non_creative, process).
is_subset_of(creation, creative).
% is_subset_of(synthesis, creation).
is_subset_of(transformation, creative).
% is_subset_of(conversion, transformation).
is_subset_of(enhancement, non_creative).
% is_subset_of(inhibition, enhancement).
% is_subset_of(promotion, enhancement).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.2.3 Ontology Constraints

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ontology_constraints.pl:
%
% Set of rules and affinities for onthological
% disambiguation of Natural Logics Grammar.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-ensure_loaded([ontology_skeleton]). % Loading ontology skeleton

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining basic affinities for prepositional phrases
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aff(in, material, material, location).
aff(in, process, material, location).
aff(in, quality, material, location).
aff(in, information, process, affiliation).
aff(in, process, quality, manner).

aff(by, process, body_part, agent).
aff(by, process, process, manner).
aff(by, process, artifact, instrument).
aff(by, process, body_fluid, agent).
aff(by, enhancement, quality, agent).

aff(of, creation, material, result).
aff(of, transformation, material, patient).
aff(of, enhancement, process, patient).
aff(of, process, body_part, beared_by).
aff(of, process, material, patient).
aff(of, process, quality, patient).
aff(of, process, information, patient).
aff(of, quality, material, beared_by).
aff(of, process, process, has_part).
aff(of, body_part, material, part_of).

aff(into, transformation, material, result).
aff(into, creation, material, direction).
aff(into, process, material, direction).

aff(from, process, body_fluid, source).
aff(from, body_fluid, body_fluid, source).
aff(from, body_fluid, body_part, source).
aff(from, process, body_part, source).

aff(at, entity, quality, condition).

aff(as, quality, entity, comparison).

aff(in_response_to, process, quality, cause).
aff(in_response_to, process, process, cause).

aff(within, entity, entity, location).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining basic affinities for compound nouns (production related)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

aff(cn, body_fluid, process, result).
aff(cn, organ, process, agent).
aff(cn, cell, process, agent).
aff(cn, process, process, manner).
aff(cn, material, quality, bearer).

aff(cn, body_part, process, bearer).
aff(cn, body_part, body_part, characterization).
aff(cn, protein, protein, characterization).
aff(cn, material, material, characterization).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.2.4 Reverse Grammar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% reverse_grammar.pl:
%
% Helper functions in order to convert parse trees
% to strings (list of words).
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-ensure_loaded([grammar]).


% reverse_np(+NP,-S) - Reverse a tree noun phrase NP into a string S
% without checking that the tree structure remains unchanged.
%
% Less accurate, but faster.
%
reverse_np(NP,S) :-% nl, nl, write("Working on: "), write(NP), nl,
                    NP = np(n(N), mod(M), ext(Ext)), lex(N, noun, _),
                    split_mod(M, Pre, Post),
                    reverse_pre(Pre, SPre), reverse_post(Post, SPost),
                    reverse_ext(Ext, SExt), append(SExt, SPost, EM),
                    append(SPre, [N], Q), append(Q, EM, S).


% reverse_np(+NP,-S) - Reverse a tree relation term into a string S.
% without checking that the tree structure remains unchanged.
%
reverse_rterm(verb(active(V),mod(M)), [V|T]) :-reverse_post(M,T), !.
reverse_rterm(verb(passive(V),mod(M)), [is,V|T]) :-reverse_post(M,T0), !,
        append(T0,[by],T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% reverse_np(+NP,-S) - Reverse a tree noun phrase NP into a string S
% and check that the tree structure remains unchanged.
%
% Accurate, but slow.
%
% Actually not used.
%
% One of the challenge appears with this approach was that trees that
% were created from nominalization were they are not always the same
% as the one generated by the parser, especially by its ontological
% analysis.
%
% Especially, a first implementation of the nominalization was at
% first giving by default an object thematic relation, but in some
% cases it should actually have been a result, making the predicate fail
% when reconstructing the tree and checking if it has changed or not.
%
% From parsing by hand:
% --> np(n(production), mod([pp(of,[PATIENT], np(n(fat),mod([]),ext([]))),
% pp(by,[agent],np(n(pancreas),mod([]),ext([]))),
%  pp(in,[location],np(n(liver),mod([]),ext([])))]),ext([]))
%
% From automated parsing:
% -->np(n(production), mod([pp(of,[RESULT],np(n(fat),mod([]),ext([]))),
%   pp(by,[agent],np(n(pancreas),mod([]),ext([]))),
%   pp(in,[location],np(n(liver),mod([]),ext([])))]),
% ext([])).
%
%
reverse_np1(NP,S) :-
          reverse_np(NP,S1), parse_np(S1,NP1), % Parse and unparse.
          reverse_np(NP1,S), parse_np(S,NP2), % Parse and unparse.
          NP = NP2, !. % Check that tree is unchanged.

parse_np(S,NP) :-np(NP,_,S,[]), !. % Return first guess of parser.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Reverse pre-modifiers.
% reverse_pre(+R, -L).
reverse_pre([],[]).
reverse_pre([adj(R)|T1],[R|T2]) :-reverse_pre(T1,T2).
reverse_pre([cn(_,n(N))|T1],[N|T2]) :-reverse_pre(T1,T2).
reverse_pre([ger(_,np(n(N), mod(M), ext([])))|T1],T2) :-
          reverse_np(np(n(N),mod(M),ext([])),S), reverse_pre(T1,Q), append(S,[s|Q],T2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Reverse post-modifiers.
reverse_post([],[]).
reverse_post([pp(P,[_],NP)|T1],T2) :-
          alignment(T1,A), reverse_np(NP,S), append(S,A,SA),
          reverse_post(T1,Q), append([P|SA],Q,T2).
reverse_post([rc(R,NP)|T1],T2) :-
          R = verb(active(V), mod(M)),
          alignment(T1,A), reverse_np(NP,S), append(S,A,SA),
          reverse_post(M, Vmod), reverse_post(T1,Q),
          T0 = [that,V|Vmod], append(T0,SA,K),
          append(K,Q,T2).
reverse_post([rc(R,NP)|T1],T) :-
          R = verb(passive(V), mod(M)),
          alignment(T1,A), reverse_np(NP,S), append(S,A,SA),
          reverse_post(M, Vmod),
          reverse_post(T1,Q), % Reverse recursively the modifiers
          T0 = [that,is,V|Vmod], append(T0,[by],T2),
          append(T2,SA,K), append(K,Q,T).
reverse_post([adv(X)|T1],[X|T]) :-
      reverse_post(T1,T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Alignment - defines by default an implicit alignment.
alignment(_,[]).
alignment(T1,T2) :-(T1 \= [] -> T2 = [and] ; T2 = []).

```

```
% Reverse extension.
reverse_ext([],[]).
reverse_ext([Ext],SExt) :-app(Ext, _, SExt, []).

% Split modifiers between pre-modifiers and post-modifiers.
split_mod([],[],[]).
split_mod([M|T],[M|T1],T2) :-is_pre(M), split_mod(T,T1,T2).
split_mod([M|T],T1,[M|T2]) :-is_post(M), split_mod(T,T1,T2).
is_pre(M) :-M = adj(_).
is_pre(M) :-M = cn(_,_).
is_pre(M) :-M = ger(_,_).
is_post(M) :-M = rc(_,_).
is_post(M) :-M = pp(_,_,_).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.3 Concept Graph Generation

C.3.1 Canonical Decomposition

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% decompose.pl:
%
% Generate a graph from natural logics proposition.
%
%
%
% NOT DO PARSING UNPARSING HERE, GIVE TREE TO OBSERVATION.
% TODO - handle prepositional verb in decomposition. [NO]
% TODO - handle predicate adjectives in decomposition. [NO]
% TODO - handle ISA with modifiers. [NO]
% TODO - handle all PC cases. [NO]
% TODO - handle adjectives for nominalization. [YES]
% TODO - should deal only with trees, except at the very end when
% creating relations -> goes back to string, otherwise creating
% errors due to ambiguities by parsing unparsing again and again.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-[grammar].   % Loading syntactic grammar.
:-ensure_loaded([reverse_grammar]). % Unparse trees.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining rules to extract information from propositions.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% decompose(S, Fs) - decompose sentence S in a list of facts Fs.
%
% Used in order to decompose a sentence into a list of facts
% whose type can be a definition, an observation, a class inclusion
% relation (isa) or a preposition relation (for instance, partonomic
% relation (in), or localisation (under)).
%
decompose(S,R) :-decompose1(S,X), flatten(X,Y), filter(Y,R), nl,
        length(R,M), write(M), write(" relations found."), nl, nl.

decompose1(S,R) :-p(Z,S,[]), % Parse proposition
        Z = [p(TNP,TVP)], TVP = vp(V,TNP2), % Get verb and noun phrases
        % nl, write("Decomposing: "), write(S),
        decompose_np(TNP,NewTNP1,T1), !, % Recursively decompose first NP
        decompose_np(TNP2,NewTNP2,T2), !, % Recursively decompose second NP
        observation(NewTNP1,V,NewTNP2,H), !, % Create observation
        append(H,T1,T2,R).

decompose1(S,R) :-p(Z,S,[]), % Parse proposition
        Z = [p(TNP,TVP)], TVP = vp(V), % Get verb and noun phrase
        % nl, write("Decomposing (special case): "), write(S), nl,
          decompose_np(TNP,NewTNP1,T), !, % Recursively decompose first NP
          observation(NewTNP1,V,H),!, % Create observation
          append(H,T,R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining rules to extract informations from NP.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

decompose_np(TNP, NewTNP, T) :-TNP = np(n(_), mod(_), ext([])),
          % write("No extension: "), write(TNP), nl,
          decompose_np1(TNP, NewTNP, T).

% Apposition of a NP.
decompose_np(TNP, NewTNP, [H|T]) :-TNP = np(n(N1), mod(M), ext([ap(TNP2)])),
          % write("Extension as AP: "), write(TNP), nl,
          NTNP = np(n(N1), mod(M), ext([])), decompose_np1(TNP2, NewTNP2, T0),
          decompose_np1(NTNP, NewTNP, T1),
          reverse_np(NewTNP2, NewSNP2), H = isa([N1], NewSNP2),
          append(T0,T1,T).

% Active isa-PC without modifiers.
decompose_np(TNP, NewTNP, [H|T]) :-TNP = np(n(N), mod(M), ext([pc(R,TNP2)])),
          % nl, nl, write("Extension as PC: "), write(TNP), nl,
          R = verb(active(isa), mod([])), decompose_np(TNP2, NewTNP2, T0),
          NTNP = np(n(N), mod(M), ext([])), decompose_np(NTNP, NewTNP, T1),
          reverse_np(NewTNP, NewSNP), reverse_np(NewTNP2, NewSNP2),
          H = isa(NewSNP, NewSNP2),
          append(T0, T1, T).

% Active PC without modifiers.
decompose_np(TNP, NewTNP, [H|T]) :-TNP = np(n(N), mod(M), ext([pc(R,TNP2)])),
          % nl, nl, write("Extension as PC: "), write(TNP), nl,
          R = verb(active(V), mod([])), decompose_np(TNP2, NewTNP2, T0),
          NTNP = np(n(N), mod(M), ext([])), decompose_np(NTNP, NewTNP, T1),
          reverse_np(NewTNP, NewSNP), reverse_np(NewTNP2, NewSNP2),
          H = fact(pc, NewSNP,[V], NewSNP2, definition),
          append(T0, T1, T).

% Active PC-isa with modifiers - TODO.
% Active PC with modifiers - TODO.
% Passive PC without modifiers - TODO.
% Passsive PC with modifiers - TODO.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% VP - Active without modifiers.
decompose_vp(TNP1, R, TNP2, NewTNP2, [H|T]) :-R = verb(active(V), mod([])),
          % write("Active VP without modifiers: "), write(V), nl,
          decompose_np(TNP2, NewTNP2, T),
          TNP1 = np(n(N), mod(_), ext([])),
          NewTNP1 = np(n(N), mod([rc(R, NewTNP2)]), ext([])),
          reverse_np(NewTNP1, SNP1), reverse_np(NewTNP2,SNP2),
          H = fact(rc,SNP1,[V],SNP2,definition), !.

% VP - Active with modifiers.
```

```
decompose_vp(TNP1, R, TNP2, NewTNP2,[H1,H2|T]) :-R = verb(active(V), mod(M1)),
          % write("Active VP with modifiers: "), write(V), write(M1), nl,
          nomi(N,V), nomi_adv(M1,M2), % Checking for nominalization.
          decompose_np(TNP2,NewTNP2,T1),
          TNP1 = np(n(N1), mod([_]), ext([])),
          NewTNP1 = np(n(N1), mod([]), ext([])),
          choice_of_relation(N, NewTNP2, OF),
          TrN = np(n(N),mod([pp(of,[OF],NewTNP2),pp(by,[agent],NewTNP1)|M2]),ext([])),
          decompose_np1(TrN,NewTrN,T2), append(T1,T2,T),
          reverse_np(NewTNP1, SNP1), reverse_np(NewTNP2, SNP2),
          reverse_np(NewTrN,SrN), reverse_rterm(R,SR),
          H1 = fact(rc,SNP1,SR,SNP2,definition), H2 = attach(H1,SrN), !.

% VP − Passive without modifiers.
decompose_vp(TNP1, R, TNP2, NewTNP2, [H|T]) :-R = verb(passive(V), mod([])),
          % write("Passive VP without modifiers: "), write(V), nl,
          decompose_np(TNP2, NewTNP2, T),
          TNP1 = np(n(N), mod(_), ext([])),
          NewTNP1 = np(n(N), mod([rc(R, NewTNP2)]), ext([])),
          reverse_np(NewTNP1,SNP1), reverse_np(NewTNP2,SNP2),
          H = fact(rc,SNP1,[is,V,by],SNP2,definition), !.

% VP − Passive with modifiers.
decompose_vp(TNP1, R, TNP2, NewTNP2, [H1,H2|T]) :-R = verb(passive(V), mod(M1)),
          % write("Passive VP with modifiers: "), write(V), nl,
          lex(V1,_,_,V,_), nomi(N,V1), nomi_adv(M1,M2), % Checking for nominalization.
          decompose_np(TNP2, NewTNP2, T1),
          TNP1 = np(n(N1), mod([_]), ext([])),
          NewTNP1 = np(n(N1), mod([]), ext([])),
          choice_of_relation(N, NewTNP2, OF),
          TrN = np(n(N),mod([pp(of,[OF],NewTNP2),pp(by,[agent],NewTNP1)|M2]),ext([])),
          decompose_np(TrN,NewTrN,T2), append(T1,T2,T),
          reverse_np(NewTNP1,SNP1), reverse_np(NewTNP2,SNP2),
          reverse_np(NewTrN,SrN), reverse_rterm(R,SR),
          H1 = fact(rc,SNP1,SR,SNP2,definition), H2 = attach(H1,SrN), !.


% This step is important, because if the relation is not chosen
% properly, the parsing might fail later on.
choice_of_relation(N1,NewTNP2,OF) :-
          % nl, write("Object relation for "), write(N1),
          % write(" and "), write(NewTNP2), nl,
(          lex(N1, noun, S1), NewTNP2 = np(n(N2), mod(_), ext(_)), lex(N2, noun, S2),
           aff(of, L1, L2, R), isa(S1,L1), isa(S2,L2)
          −> OF = R ; write("Default of relation"), OF = patient).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Basic case − without any modifiers.
decompose_np1(TNP, NewTNP, []) :-TNP = np(n(_), mod([]), ext([])),
          % write("Basic case: "), write(TNP), nl,
          NewTNP = TNP, !.

% With only one modifier − RC.
decompose_np1(TNP, NewTNP, [H|T]) :-TNP = np(n(N), mod([rc(V,TNP2)]), ext([])),
          % nl, write("Decomposing as leaf RC:"), write(TNP), nl,
          decompose_vp(TNP, V, TNP2, NewTNP2, T),
          NewTNP = np(n(N), mod([rc(V,NewTNP2)]), ext([])),
          reverse_np(NewTNP, NewSNP),
          H = isa(NewSNP,[N]), !.

% With only one modifier - PP.
decompose_np1(TNP, NewTNP, [H1,H2|T]) :-TNP = np(n(N),mod([pp(PREP,[R],TNP2)]),ext([])),
          % nl, write("Decomposing as leaf PP: "), write(TNP), nl,
          decompose_np(TNP2, NewTNP2, T),
          NewTNP = np(n(N), mod([pp(PREP,[R],NewTNP2)]), ext([])),
          reverse_np(NewTNP, NewSNP), reverse_np(TNP2, NewSNP2),
          H1 = fact(prep, NewSNP, PREP, NewSNP2, R), H2 = isa(NewSNP, [N]), !.

% With only one modifier - CN.
decompose_np1(TNP, NewTNP, [H1,H2]) :-TNP = np(n(N), mod([cn([R],n(N2))]), ext([])),
          % nl, write("Decomposing as leaf CN: "), write(TNP), nl,
          NewTNP = TNP,
          reverse_np(TNP, NewSNP),
          H1 = isa(NewSNP,[N]), H2 = fact(cn, NewSNP, none, [N2], [R]), !.

% With only one modifier - GER.
decompose_np1(TNP, NewTNP, [H1,H2|T]) :-TNP = np(n(N), mod([ger(R,TNP2)]), ext([])),
          % nl, write("Decomposing as leaf GER: "), write(TNP), nl,
          decompose_np(TNP2, NewTNP2, T),
          NewTNP = np(n(N), mod([ger(R,NewTNP2)]), ext([])), !,
          reverse_np(NewTNP, NewSNP), reverse_np(NewTNP2, NewSNP2),
          H1 = fact(ger, NewSNP, none, NewSNP2, R), H2 = isa(NewSNP,[N]), !.

% With only one modifier - ADJ - intersective.
decompose_np1(TNP, NewTNP, [H1,H2]) :-TNP = np(n(N), mod([adj(A)]), ext([])),
          lex(A, adj, inter, _, _),
          % nl, write("Decomposing as leaf ADJ: "), write(TNP), nl,
          NewTNP = TNP,
          reverse_np(TNP, NewSNP),
          H1 = isa(NewSNP, [N]), H2 = isa(NewSNP, [A, entity]), !.

% With only one modifier - ADJ - subsective.
decompose_np1(TNP, NewTNP, [H1]) :-TNP = np(n(N), mod([adj(A)]), ext([])),
          lex(A, adj, subs, _, _),
          % nl, write("Decomposing as leaf ADJ: "), % write(TNP), nl,
          NewTNP = TNP,
          reverse_np(TNP, NewSNP),
          H1 = isa(NewSNP, [N]), !.

% With only one modifier - ADJ - non-subsective or privative
decompose_np1(TNP, NewTNP, []) :-TNP = np(n(_), mod([adj(_)]), ext([])),
          % nl, write("Decomposing as leaf ADJ: "), write(TNP), nl,
          NewTNP = TNP, !.

% General case - with many modifiers.
decompose_np1(TNP, NewTNP, T) :-TNP = np(n(N), mod(M), ext([])), M \= [],
          % nl, write("General case: "), write(TNP), nl,
          dec_modifiers(N,M,M2,R), % Remove isa relations recursively.
          NewTNP = np(n(N), mod(M2), ext([])), % Create the new main tree.
          nsubsets(M2, SubMs), !, % Create new subsets of modifiers.
          convert_np(N, SubMs, SubTNPs), !, % Convert them into proper trees.
          reverse_np(NewTNP, NewSNP), % Transform main tree to string.
          create_n_isa(NewSNP,SubTNPs, R1),!, % Create isa relations.
```

```
        decompose_subnp(N, SubMs, R2), % Decompose new trees recursively.
        append(R,R1,R2,T). % Append lists.

decompose_np1(X,_,[]) :-write("default case for: "), write(X), nl.

% FIX: Problem: list that is not flat, use of head here cause append crashes.
% dec_modifiers(+N,+L1,−L2,−R).
dec_modifiers(_,[],[],[]).
dec_modifiers(N, [H0|T0],[H1|T1], [R0|T2]) :-TNP = np(n(N), mod([H0]), ext([])),
        decompose_np(TNP, NewTNP, R0), NewTNP = np(n(N), mod([H1]), ext([])),
        dec_modifiers(N,T0,T1, T2), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining rules to create observation based on type of relation term.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ISA + modifiers is not handled for now properly. Only the case with
% one PP.

% Case: ISA.
observation(TNP1,verb(active(isa),mod([])),TNP2,[H]) :-
        reverse_np(TNP1,SNP1), reverse_np(TNP2,SNP2),
        H = isa(SNP1,SNP2),!.

% Case: Active verb.
observation(TNP1,verb(active(V), mod([])),TNP2,[H]) :-% Without prepositional term.
        % nl, write("Observation active without modifiers"), nl,
        reverse_np(TNP1,SNP1), reverse_np(TNP2,SNP2),
        H = fact(prop,SNP1,[V],SNP2, observation), !. % Create observation.
observation(TNP1,R,TNP2,[H1,H2|T]) :-% With prepositional terms.
        R = verb(active(V), mod(M1)), V \= isa,
        % nl, write("Observation active with modifiers"), nl, !,
        nomi(N,V), nomi_adv(M1,M2),
        choice_of_relation(N,TNP2,OF),
        TrN = np(n(N), mod([pp(of,[OF],TNP2), pp(by,[agent],TNP1)|M2]), ext([])), !,
        decompose_np1(TrN,NewTrN,T), reverse_rterm(R,SR), !,
        reverse_np(TNP1,SNP1), reverse_np(TNP2,SNP2), reverse_np(NewTrN,SrN),
        H1 = fact(prop,SNP1,SR, SNP2, observation), H2 = attach(H1,SrN), !.

% Case: Passive verb with agent.
observation(TNP1,verb(passive(V), mod([])), TNP2,[H]) :-% Without prepositional term.
        reverse_np(TNP1,SNP1), reverse_np(TNP2,SNP2),
        H = fact(observation,SNP1,[is,V,by],SNP2), !. % Create observation.
observation(TNP1,R,TNP2,[H1,H2|T]) :-% With prepositional terms.
        R = verb(passive(V), mod(M1)),
        lex(V1,_,_,V,_), nomi(N,V1), nomi_adv(M1,M2),
        choice_of_relation(N,TNP1,OF),
        TrN = np(n(N), mod([pp(of,[OF],TNP1), pp(by,[agent],TNP2)|M2]), ext([])),
        decompose_np(TrN,NewTrN,T), reverse_rterm(R,SR),
        reverse_np(TNP1,SNP1), reverse_np(TNP2,SNP2), reverse_np(NewTrN,SrN),
        H1 = fact(prop,SNP1,SR,SNP2, observation), H2 = attach(H1,SrN), !.

% Default case.
observation(_,_,_,[]) :-nl, write("Default observation."), nl.
```

```

% Case: Passive verb without agent.
observation(TNP1,verb(passive(V), mod([])),[H]) :-% Without prepositional term.
          reverse_np(TNP1,SNP1),
          H = fact(prop,SNP1,[is,V,by],[entity],observation), !. % Create observation.
observation(TNP1,R,[H1,H2|T]) :-% With prepositional terms.
          R = verb(passive(V), mod(M1)),
          lex(V1,_,_,V,_), nomi(N,V1), nomi_adv(M1,M2),
          choice_of_relation(N,TNP1,OF),
          TrN = np(n(N), mod([pp(of,[OF],TNP1)|M2]), ext([])),
          decompose_np(TrN,NewTrN,T), reverse_rterm(R,SR),
          reverse_np(TNP1, SNP1), reverse_np(NewTrN,SrN),
          H1 = fact(prop,SNP1,SR,[entity], observation), H2 = attach(H1,SrN), !.

% Default case.
observation(_,_,[]) :-nl, write("Default observation."), nl.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Helper functions
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Append three lists together.
append([],[],L,L).
append([],[H|T],L,[H|R]) :-append([],T,L,R).
append([H|T],L0,L1,[H|R]) :-append(T,L0,L1,R).

% Filter duplicates in a list.
filter([], []).
filter([H|T], [H|T1]) :-subtract(T,[H],T2), filter(T2, T1).

% Convert a list of nouns(N) and modifiers(M) into a list of trees.
convert_np(_,[],[]).
convert_np(N,[M|T1],[TNP|T2]) :-TNP = np(n(N),mod(M),ext([])),
          convert_np(N,T1,T2).

% Recursively decomposing a list of NPs trees.
decompose_subnp(_,[],[]).
decompose_subnp(N,[HL|TL],R):-
          TNP = np(n(N), mod(HL), ext([])), decompose_np(TNP,_,R1), !,
          decompose_subnp(N,TL,R2), append(R1,R2,R).

% Create n ISA relations based on a list of ISA relations: tree form
create_n_isa(_,[],[]).
create_n_isa(SNP1,[H1],[H2]) :-reverse_np(H1,SNP2), H2 = isa(SNP1, SNP2).
create_n_isa(SNP1,[H1|T1],[H2|T2]) :-reverse_np(H1,SNP2),
          H2 = isa(SNP1, SNP2), create_n_isa(SNP1,T1,T2).

% Transform adverbials modifiers into concept modifiers.
nomi_adv([],[]).
nomi_adv([adv(H1)|T1],[adj(H2)|T2]) :-adv_adj(H1,H2), nomi_adv(T1,T2).
nomi_adv([H|T1],[H|T2]) :-H = pp(_,_,_), nomi_adv(T1,T2).



```

```
% nsubsets(+X,-R) - Returns the n subsets R of length n-1 of the list X
% of size n.
%
% Example of use:
% ?- nsubsets([a,b,c],X).
% X = [[a, b], [a, c], [b, c]].
%
nsubsets([],[]).
nsubsets(X,R) :-length(X,L), N is L-1, setof(Y,subset(N,X,Y),R).
subset(0,[],[]).
subset(L,[E|T1],[E|T2]):- succ(PL,L),(PL>0->subset(PL,T1,T2); T2=[]).
subset(L, [_|T1], T2):- subset(L,T1,T2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.3.2 Inference Engine

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% infer.pl:
%
% Defining inference rules working over our facts definition.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-dynamic fact/4.
:-dynamic fact/5.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining the main predicates.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% infer_bis(+KB,-I) - returns all possible inferences I from the list
% of facts KB given in input.
%
infer(KB,I) :-
        infer_isa(KB,Z), append(KB,Z,X),
        infer_fact(X,Y), append(X,Y,W),
        filter(W,I).

% infer_isa(+KB,-I) - returns all inferences on isa relations from the
% list of facts KB given in input.
%
infer_isa(KB,I) :-setof(isa(C,D), inf_rule(KB,C,D),I), !.
infer_isa(_,[]). % In case of empty KB or no relation inferred.

% infer_fact(+KB,-I) - returns all inferences on facts from the list
% of facts KB given in input.
%
infer_fact(KB,I) :-setof(fact(T,C,V,D,R), inf_rule(KB,T,C,V,D,R),I),!.
infer_fact(_,[]). % In case of empty KB or no relation inferred.

% subclass(+KB,?C,?D) - returns true if C isa a subclass of D in KB.
%
subclass(KB,C,D) :-subclass(KB,C,D,[]).
subclass(KB,X,Y,V) :-member(isa(X,Y),KB), \+ memberchk((X,Y),V).
subclass(KB,X,Y,V) :-member(isa(X,Z),KB), \+ memberchk((X,Z),V),
        subclass(KB,X,Y,[(X,Y)|V]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining inference rules on facts and class inclusion.
% Domain specific inference rules are not considered here.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Rule 1:
```

```
% If (D isa D") & (C R D) then (C R D").
%
inf_rule(KB,T,C,V,Dsup,R) :-
        subclass(KB,D,Dsup),
        member(fact(T,C,V,D,R), KB).

% Rule 2:
% If (C" isa C) & (C R D) then (C" R D).
%
inf_rule(KB,T,Csub,V,D,R) :-
        subclass(KB,Csub,C),
        member(fact(T,C,V,D,R), KB).

% Rule 3:
% If (C" isa C) & (D isa D") & (C R D) then (C" R D").
%
inf_rule(KB,T,Csub,V,Dsup,R) :-
        subclass(KB,Csub,C), subclass(KB,D,Dsup),
        member(fact(T,C,V,D,R), KB).

% Rule 4:
% If (C" isa C) & (C isa D) then (C" isa D).
%
inf_rule(KB,Csub,D) :-subclass(KB,Csub,C), member(isa(C,D), KB).

% Rule 5:
% If (C isa D) & (D isa D") then (C isa D").
%
inf_rule(KB,C,Dsup) :-subclass(KB,D,Dsup), member(isa(C,D), KB).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.3.3 Subsumption Engine

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% subsum.pl:
%
% Defining subsumption rules working over our facts definition.
%
% The ranking algorithm has been implemented and is expected to work
% properly. Other parts of the algorithm should be considered as draft
% for a future implementation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-ensure_loaded([grammar]).
:-ensure_loaded([reverse_grammar]).
:-ensure_loaded([infer]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Computing rank of concepts.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Compute ranks of all concepts.
allRank(KB,L) :-rk(KB,[],[],0,L).

% rk(+KB,+AlreadySeen,+PrevRanks,-CurrentRank,-Rank).
rk(KB,[],[],0,Acc) :-gRank(KB,[],L,[],0), L \= [], H = (0,L),
        M is 1, rk(KB,L,[H],M,Acc), !.
rk(KB,S,T,N,Acc) :-N > 0, gRank(KB,S,L,T,N), L \= [], H = (N,L), M is N+1,
        append(S,L,Seen),rk(KB,Seen,[H|T],M,Acc), !.
rk(KB,_,T,N,Acc) :-N > 0, gRank(KB,_,L,T,N), L == [], T = Acc.

% Returns list of concepts L in the KB of rank R.
gRank(KB,Seen,L,Prev,R) :-my_setof(C,rank(KB,Seen,C,Prev,R),L), !.

% rank(+KB,+C,-R) - Returns rank R of concept C in KB.
rank(KB,[],C,[],0) :-count(KB,C,_,0). % write(C), write(" is ranked 0"), nl.
rank(KB,Seen,C,Prev,R) :-R > 0, member(fact(_,C,_,D,_),KB),
        \+ member(C,Seen), N is R-1, concepts_of_rank(Prev,N,L), member(D,L).
        % write(C), write(" is ranked "), write(R), nl.

concepts_of_rank([H|T],R,L) :-(H = (R,L) -> true; concepts_of_rank(T,R,L)), !.
concepts_of_rank(_,_,[]).

% count(+KB,-C,-N) - Returns the number N of non isa relation in the KB
% for concept C. NB: Using findall + sort instead of setof because they
% return an empty list if 0 whereas setof return false.
%
count(KB,C,Z,N) :-concept(KB,C),
        my_setof(fact(T,C,R,D,V),member(fact(T,C,R,D,V),KB),Z),
        length(Z,N).


```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Helper functions.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% concepts(+KB,-C) - Returns the list of all concepts defined in the KB.
%
concepts(KB,C) :-setof(X,concept(KB,X),C).
concept(KB,C) :-member(fact(_,C,_,_,_),KB).
concept(KB,C) :-member(fact(_,_,_,C,_),KB).
concept(KB,C) :-member(isa(C,_),KB).
concept(KB,C) :-member(isa(_,C),KB).

% my_setof(Obj, Goal, List) - Returns the set of all
my_setof(Obj, Goal, List) :-(\+ Goal -> List = [] ; setof(Obj,Goal,List)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.4 Graph Search Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Search.pl:
%
%  Graph Search for Natural Logics KB.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-dynamic fact/5. % Create and return facts.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% IDDFS (iterative deepening depth first search).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% search(+KB,+D,+S,-A) - predicate querying the knowledge base KB
% using IDDFS with a maximum depth D, with S as start state and G
% as goal state in order to return the answer A.
%
% This algorithm tries frist to find paths without reverted edges.
%
% Example:
% ?- search(KB,4,[insulin],R).
% R =[[insulin], which, isa,
% [peptide, hormone, that, is, produced, by, betacell, of,
% pancreatic, islet]] R = [insulin, which, which, isa, protein].
%
search(KB,D,First,[every|R]) :-
        iddfs(KB,D,0,First,_,R,false), R \= [First].
search(KB,D,First,[some|R]) :-
        iddfs(KB,D,0,First,_,R,true), R \= [First].


% search(+KB,+D,+S,+G,-A) - predicate querying the knowledge base KB
% using IDDFS with a maximum depth D, with S as start state and G
% as goal state in order to return the answer A.
%
% This algorithm tries frist to find paths without reverted edges.
%
% Example:
% ?- search(KB,4,[insulin],[protein],R).
% R = [insulin, which, isa, hormone, which, isa, protein].
%
search(KB,D,First,Goal,[every|R]) :-
        iddfs(KB,D,0,First,Goal,R,false).
search(KB,D,First,Goal,[some|R]) :-
        iddfs(KB,D,0,First,Goal,R,true).
search(_,_,F,G,R) :-
        R = [F, and, G, are, apparently, not, connected].


```

```
iddfs(KB,MaxD,D,F,G,R,A) :-
          D < MaxD, dls(KB,D,F,G,Path,R,A),
          write("Current depth: "), write(D), nl,
          write("Nodes: "), write(Path), nl.
iddfs(KB,MaxD,D,F,G,R,A) :-
          D < MaxD, NewDeep is D+1,
          iddfs(KB,MaxD,NewDeep,F,G,R,A).

% dls(+KB,+D,+F,+G,-P,-R,-A) - Depth limited DFS algorithm for the
% knowledge base KB, with a depth limit D, starting from F until the
% goal G, returning then the answer R for the path P. The parameter A,
% indicates if reversed arcs are allowed.
%
dls(_,_,G,G,[G],[G],_).
dls(KB,D,F,G,[G|Path],R,A) :-
          0 < D, Ds is D - 1,
          dls(KB,Ds, F,OneButLast,Path,H,A),
          arc(KB,OneButLast,G,[I1,I2],A),
          \+member(G,Path),
(          Path = [F] -> append(H,I1,R) ; append(H,I2,R)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining direct arcs for the graph search.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% arc(+KB,+S,+G,-I,-R) - arc between states S and G contained in the
% knowledge base KB and carrying the information I. R indicates if the
% arc is reverted or not.
%
% These graphs are either directly deduced from the knowledge
% (observation, definition, isa), but uses the thematic relations from
% the ontological analysis (agent, location, ...) to tailor a more
% specific answer.
%
% PPs involving "in_response_to", "at" and "as" are not represented here
% since they only define comparison, conditions and cause which are not
% handled by the graph search.
%
arc(KB,S1,S2,[I1,I2],_) :-% Isa relations.
          member(isa(S1,S2),KB),
          I1 = [is,a,S2],
          I2 = [which,is,a,S2].


arc(KB,S1,S2,[I1,I2],_) :-% Observations and definitions.
          member(fact(_,S1,V,S2,T),KB),
          member(T,[def,observation]),
          I1 = [V, by, T, S2],
          I2 = [which, V, by, T, S2].


arc(KB,S1,S2,[I1,I2],_) :-% PP: in - location: cell in pancreas.
          member(fact(prep,S1,in,S2,location),KB),
        I1 = [is,located,in,S2],
        I2 = [which,is,located,in,S2].
arc(KB,S1,S2,[I1,I2],false) :-% PP: in – affiliation TODO
        member(fact(prep,S1,in,S2,affiliation),KB),
        I1 = [is, related, to, S2],
        I2 = [which,is,related,to,S2].
arc(KB,S1,S2,I,false) :-% PP: in – manner TODO
        member(fact(prep,S1,in,S2,manner),KB),
        I = [which,is,todo,S2].


arc(KB,S1,S2,[I1,I2],_) :-% PP: by – agent: production by pancreas.
        member(fact(prep,S1,by,S2,agent),KB),
        I1 = [made, by, S2],
        I2 = [which,is,made,by,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: by – instrument: payment by credit card.
        member(fact(prep,S1,by,S2,instrument),KB),
        I1 = [is,made,using,S2],
        I2 = [which,is,made,using,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: by – manner: production by glycogenesis.
        member(fact(prep,S1,by,S2,manner),KB),
        I1 = [is,made,by,means,of,S2],
        I2 = [which,is,made,by,means,of,S2].


arc(KB,S1,S2,[I1,I2],_) :-% PP: of – result: concentration of glucose.
        member(fact(prep,S1,of,S2,beared_by),KB),
        I1 = [is,beared,by,S2],
        I2 = [which,is,beared,by,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: of – result: synthesis of production.
        member(fact(prep,S1,of,S2,has_part),KB),
        I1 = [has,part,S2],
        I2 = [which,has,part,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: of – result: synthesis of production.
        member(fact(prep,S1,of,S2,part_of),KB),
        I1 = [is,part,of,S2],
        I2 = [which,is,part,of,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: of – patient: conversion of glucose.
        member(fact(prep,S1,of,S2,patient),KB),
        I1 = [involves,S1],
        I2 = [which,involves,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: of – result: production of glucose.
        member(fact(prep,S1,of,S2,result),KB),
        I1 = [creates,S2],
        I2 = [which,creates,S2].


% TODO FROM HERE.

arc(KB,S1,S2,[I1,I2],_) :-% PP: into – source: production into blood.
        member(fact(prep,S1,into,S2,direction),KB),
        I1 = [is,made,in,the,direction,of,S2],
        I2 = [which,is,made,in,the,direction,of,S2].
arc(KB,S1,S2,[I1,I2],_) :-% PP: into – result: transformation into glucose.
        member(fact(prep,S1,into,S2,result),KB),
        I1 = [creates, S2],
        I2 = [which,creates,S2].
```

```


arc(KB,S1,S2,[I1,I2],_) :-% PP: from − origin: glucose from blood.
          member(fact(prep,S1,from,S2,origin),KB),
          I1 = [comes,from,S2],
          I2 = [that,comes,from,S2].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

arc(KB,S1,S2,[I1,I2],_) :-% CN: agent
          member(fact(cn,S1,none,S2,agent),KB),
          I1 = [initiates,S2],
          I2 = [which,initiates,S2].

arc(KB,S1,S2,[I1,I2],_) :-% CN: bearer
          member(fact(cn,S1,none,S2,bearer),KB),
          I1 = [bears,S2],
          I2 = [which,bears,S2].

arc(KB,S1,S2,[I1,I2],_) :-% CN: agent
          member(fact(cn,S1,none,S2,location),KB),
          I1 = [is,located,in,S2],
          I2 = [which,is,located,in,S2].

arc(KB,S1,S2,[I1,I2],_) :-% CN: patient
          member(fact(cn,S1,none,S2,patient),KB),
          I1 = [is,affected,by,S2],
          I2 = [which,is,affected,by,S2].

arc(KB,S1,S2,[I1,I2],_) :-% CN: result
          member(fact(cn,S1,none,S2,result),KB),
          I1 = [results,from,S2],
          I2 = [which,results,from,S2].

arc(KB,S1,S2,[I1,I2],_) :-% CN: default
          member(fact(cn,S1,none,S2,R),KB),
          I1 = [is,related,by,R,to,S2],
          I2 = [which,is,related,by,R,to,S2].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% TO DETAIL
arc(KB,S1,S2,[I1,I2],_) :-% GER: default
          member(fact(ger,S1,none,S2,R),KB),
          I1 = [is,related,by,R,to,S2],
          I2 = [which,is,related,by,R,to,S2].



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Defining indirect arcs for the graph search.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


```

```
arc(KB,S1,S2,[I1,I2],true) :-% ISA relation.
        member(isa(S2,S1),KB),
        I1 = [are, a, S2],
        I2 = [whereof,some,are,a,S2].

arc(KB,S1,S2,[I1,I2],true) :-
        member(fact(_,S2,V,S1,T),KB), % Passivation of the form.
        member(T,[def,observation]),
        I1 = [V, by, T, S2],
        I2 = [whereof,by,T,some,are,V,d,by,S2].

arc(KB,S1,S2,[I1,I2],true) :-% PP: in – location: cell in pancreas.
        member(fact(prep,S2,in,S1,location),KB),
        I1 = [is,the,host,of,S2],
        I2 = [which,is,the,host,of,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: in – affiliation TODO
        member(fact(prep,S2,in,S1,affiliation),KB),
        I1 = [which,is,related,to,S2],
        I2 = [which,is, related,to,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: in – manner TODO
        member(fact(prep,S2,in,S1,manner),KB),
        I1 = [is,todo,S2],
        I2 = [which,is,todo,S2].

arc(KB,S1,S2,[I1,I2],true) :-% PP: of – result: concentration of glucose.
        member(fact(prep,S2,of,S1,beared_by),KB),
        I1 = [bears, S2],
        I2 = [which,bears,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: of – result: synthesis of production.
        member(fact(prep,S2,of,S1,has_part),KB),
        I1 = [is,part,of,s2],
        I2 = [which,is,part,of,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: of – result: synthesis of production.
        member(fact(prep,S2,of,S1,part_of),KB),
        I1 = [has,part,S2],
        I2 = [which,has,part,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: of – patient: conversion of glucose.
        member(fact(prep,S2,of,S1,patient),KB),
        I1 = [which,is,affected,in,S2],
        I2 = [which,is,affected,in,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: of – result: production of glucose.
        member(fact(prep,S2,of,S1,result),KB),
        I1 = [is,created,by,S2],
        I2 = [which,is,created,by,S2].


arc(KB,S1,S2,[I1,I2],true) :-% PP: into – source: glucose into blood.
        member(fact(prep,S2,into,S1,direction),KB),
        I1 = [is,the,destination,of,S2],
        I2 = [which,is,the,destination,of,S2].
arc(KB,S1,S2,[I1,I2],true) :-% PP: into – result: transformation into glucose.
        member(fact(prep,S2,into,S1,result),KB),
        I1 = [originates,from,S2],
        I2 = [which,originates,from,S2].


arc(KB,S1,S2,[I1,I2],true) :-% PP: from – source: glucose from blood.
```

```
        member(fact(prep,S2,from,S1,source),KB),
        I1 = [is,the,source,of,S2],
        I2 = [which,is,the,source,of,S2].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

arc(KB,S1,S2,[I1,I2],true) :-% CN: agent
        member(fact(cn,S2,none,S1,agent),KB),
        I1 = [is,initiated,by,S1],
        I2 = [which,is,initiated,by,S1].

arc(KB,S1,S2,[I1,I2],true) :-% CN: bearer
        member(fact(cn,S2,none,S1,bearer),KB),
        I1 = [is,beared,by,S1],
        I2 = [which,is,beared,by,S1].

arc(KB,S1,S2,[I1,I2],true) :-% CN: agent
        member(fact(cn,S2,none,S1,location),KB),
        I1 = [is,the,location,of,S1],
        I2 = [which,is,the,location,of,S1].

arc(KB,S1,S2,[I1,I2],true) :-% CN: patient
        member(fact(cn,S2,none,S1,patient),KB),
        I1 = [affects,S1],
        I2 = [which,affects,S1].

arc(KB,S1,S2,[I1,I2],true) :-% CN: result
        member(fact(cn,S2,none,S1,result),KB),
        I1 = [creates,S1],
        I2 = [which,creates,S1].

arc(KB,S1,S2,[I1,I2],true) :-% CN: default
        member(fact(cn,S2,none,S1,R),KB),
        I1 = [is,related,by,R,to,S1],
        I2 = [which,is,related,by,R,to,S1].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

arc(KB,S1,S2,[I1,I2],true) :-% GER: default.
        member(fact(ger,S2,none,S1,R),KB),
        I1 = [is,related,by,R,to,S2],
        I2 = [which,is,related,by,R,to,S2].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C.5 Unit tests

The following unit tests aim at showing the basic functionalists handled by each script. On top of their use in order to test the non-regression of the features during the implementation, these unit tests show examples that can be used by the reader to understand the exact behaviour of the function being tested.

C.5.1 Test of Extended Grammar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% test_grammar.pl:
%
%  Unit tests for extended grammar.
%
%
%
% This file aims at testing the basic features that should be handled by
% Natural Logics grammar.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-ensure_loaded(grammar_ambiguous). % Loading the extended grammar.
:-ensure_loaded(lexicon_basic_ambiguous). % Loading the example lexicon.


:-begin_tests(grammar).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of construction rules for pre and post-modifiers.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Use of pre-modifiers: adjectives.
test(pre_adj_1) :-np(_,[synchronous,production],[]). % Use of an adjectives.
test(pre_adj_2) :-np(_,[synchronous,former,production],[]). % Use of multiple adj.
test(pre_adj_3, fail) :-np(_,[synchronous,and,former,production],[]). % No keyword and.

% Use of pre-modifiers: compound nouns.
test(pre_cn_1) :-np(_,[glucose,production],[]). % Normal compound noun with 2 words.
test(pre_cn_2) :-np(_,[pancreas,glucose,production],[]). % Compound noun with 3 words.
test(pre_cn_3, fail) :-np(_,[produce,production],[]). % Compound noun with verb.

% Use of pre-modifiers: possessives
test(pre_cn_1) :-np(_,[pancreas,s,production],[]). % Normal possessive with noun.
test(pre_cn_2, fail) :-np(_,[produce,s,production],[]). % Possessive with verb.
test(pre_cn_3) :-np(_,[glucose,production,s,behaviour],[]). % Possessive with comp noun.

% Use of post-modifiers: PP.
test(post_pp_1) :-np(_,[production,of,glucose],[]). % Normal use of PP.
test(post_pp_2, fail) :-np(_,[production,af,glucose],[]). % PP with wrong preposition.
test(post_pp_3, fail) :-np(_,[production,of,produce],[]). % PP with verb after prep.
test(post_pp_4) :-np(_,[production,of,glucose,in,pancreas],[]).

% Use of post−modifiers: RC.
test(post_rc_1) :-np(_,[pancreas,that,produce,glucose],[]). % Normal use of RC.
test(post_rc_2, fail) :-np(_,[pancreas,that,glucose],[]). % Wrong use of RC.
test(post_rc_3, fail) :-np(_,[cell,that,isa,betacell],[]).
test(post_rc_4, fail) :-np(_,[cell,that,isa,in,pancreas,betacell],[]).
test(post_rc_5) :-np(_,[cell,that,produce,in,pancreas,glucose],[]).
test(post_rc_6) :-np(_,[insulin,that,is,produced,by,betacell],[]).
test(post_rc_7) :-np(_,[insulin,that,is,produced,in,pancreas,by,betacell],[]).
test(post_rc_8) :-np(_,[insulin,that,is,produced],[]).
test(post_rc_9) :-np(_,[insulin,that,is,produced,in,pancreas],[]).
test(post_rc_10) :-np(_,[pancreas,that,produce,glucose,and,that,produce,insulin],[]).

% Use of post−modifiers: PC.
test(post_pc_1) :-np(_,[pancreas,",",which,produce,glucose,","],[]).
test(post_pc_2, fail) :-np(_,[pancreas,",",which,produce,","],[]).

% Use of post−modifiers: Appositions.
test(post_ap_1) :-np(_,[pancreas,",",an,insulin,","],[]).
test(post_ap_2) :-np(_,[pancreas,",",a,insulin,","],[]).
test(post_ap_3, fail) :-np(_,[pancreas,",",a,insulin],[]).

% Syntax order of modifiers:
test(mod1) :-np(_,[pancreas,s,synchronous,glucose,production,of,glucose,and,
                    that,produce,glucose],[]).
test(mod2, fail) :-np(_,[pancreas,s,glucose,synchronous,production,of,glucose],[]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of syntactic rules related to verb usage.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Respect of transitivity in the main proposition.
test(trans_1, fail) :-p([_],[doctor,smile],[]). % No use of intransitive verb.
test(trans_2, fail) :-p([_],[doctor,observe],[]). % No use of trans verb without object.
test(trans_3) :-p([_],[doctor,observe,pancreas],[]). % Normal use of transitive verb.
test(trans_4, fail) :-p([_], [doctor,smile,pancreas],[]). % No use of intrans as trans.

% Respect of transitivity in RC.
test(trans_rc_1, fail) :-np(_,[pancreas,that,smile],[]).
test(trans_rc_2, fail) :-np(_,[pancreas,that,produce],[]).
test(trans_rc_3) :-np(_,[pancreas,that,produce,insulin],[]).
test(trans_rc_4, fail):- np(_,[pancreas,that,smile,insulin],[]).

% Ability to use passive forms, even without object (agent).
test(pass_1) :-p([_],[insulin,is,produced,by,pancreas],[]).
test(pass_2) :-p([_],[insulin,is,produced,in,pancreas,by,pancreas],[]).
test(pass_3) :-p([_],[insulin,is,produced],[]).
test(pass_4) :-p([_],[insulin,is,produced,in,pancreas],[]).

% Respect of predicativity for adjectives.
test(adj_pred_1) :-p([_],[doctor,isa,young],[]).
test(adj_pred_2) :-p([_],[doctor,isa,young,red],[]).
test(adj_pred_3, fail) :-p([_],[doctor,isa,former],[]). % Non predicate adjective.
test(adj_pred_4, fail) :-p([_],[doctor,produce,former],[]). % Non copular verb.
test(adj_pred_5, fail) :-p([_],[doctor,produce,young],[]). % Non copular verb.

% Respect of rules regarding adverbs.
test(adv_1) :-p([_],[betacell,produce,in,pancreas,insulin],[]).
test(adv_2) :-p([_],[betacell,produce,synchronously,insulin],[]).
test(adv_3) :-p([_],[betacell,produce,synchronously,in,pancreas,insulin],[]).
test(adv_4, fail) :-% Wrong order of adverbs and adverbial PPs
        p([_],[betacell,produce,in,pancreas,synchronously,insulin],[]).
test(adv_5, fail) :-% Wrong position of the adverb.
        p([_],[betacell,synchronously,produce,insulin],[]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of syntactic rules related to proposition extensions.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Respect of plural readings.
test(plural_1, Z==2) :-p(X,[alphacell,and,betacell,produce,insulin],[]),
                    length(X,Z). % Distributive reading on subject.
test(plural_2, Z==2) :-p(X,[alphacell,produce,insulin,and,glucose],[]),
                    length(X,Z). % Distributive reading on object
test(plural_3, Z==4) :-p(X,[alphacell,and,betacell,produce,insulin,and,glucose],[]),
                    length(X,Z). % Double distributive reading on subject and object.
test(plural_4, fail) :-p(_,[alphacell,or,betacell,produce,insulin],[]). % Not handled.
test(plural_5, fail) :-p(_,[alphacell,produce,insulin,or,glucose],[]). % No supremum.
test(plural_6) :-p([_],[insulin,is,produced,by,alphacell,or,betacell],[]). % Supremum.


:-end_tests(grammar).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results:

```
?- run_tests.
% PL-Unit: grammar
Warning: c:/.../test_grammar.pl:29:
        PL-Unit: Test pre_adj_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:30:
        PL-Unit: Test pre_adj_2: Test succeeded with choicepoint
.
Warning: c:/.../test_grammar.pl:34:
        PL-Unit: Test pre_cn_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:35:
        PL-Unit: Test pre_cn_2: Test succeeded with choicepoint
.
Warning: c:/.../test_grammar.pl:39:
        PL-Unit: Test pre_cn_1: Test succeeded with choicepoint
.
Warning: c:/.../test_grammar.pl:41:
        PL-Unit: Test pre_cn_3: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:44:
        PL-Unit: Test post_pp_1: Test succeeded with choicepoint
..
Warning: c:/.../test_grammar.pl:47:
        PL-Unit: Test post_pp_4: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:50:
        PL-Unit: Test post_rc_1: Test succeeded with choicepoint
...
Warning: c:/.../test_grammar.pl:54:
        PL-Unit: Test post_rc_5: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:55:
        PL-Unit: Test post_rc_6: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:56:
        PL-Unit: Test post_rc_7: Test succeeded with choicepoint
ERROR: c:/.../test_grammar.pl:57:
        test post_rc_8: failed

ERROR: c:/.../test_grammar.pl:58:
        test post_rc_9: failed

Warning: c:/.../test_grammar.pl:59:
        PL-Unit: Test post_rc_10: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:62:
        PL-Unit: Test post_pc_1: Test succeeded with choicepoint
.
Warning: c:/.../test_grammar.pl:66:
        PL-Unit: Test post_ap_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:67:
        PL-Unit: Test post_ap_2: Test succeeded with choicepoint
.
Warning: c:/.../test_grammar.pl:71:
        PL-Unit: Test mod1: Test succeeded with choicepoint
...
Warning: c:/.../test_grammar.pl:85:
        PL-Unit: Test trans_3: Test succeeded with choicepoint
...
Warning: c:/.../test_grammar.pl:91:
        PL-Unit: Test trans_rc_3: Test succeeded with choicepoint
.
```

```
Warning: c:/.../test_grammar.pl:95:
        PL−Unit: Test pass_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:96:
        PL−Unit: Test pass_2: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:97:
        PL−Unit: Test pass_3: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:98:
        PL−Unit: Test pass_4: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:101:
        PL−Unit: Test adj_pred_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:102:
        PL−Unit: Test adj_pred_2: Test succeeded with choicepoint
...
Warning: c:/.../test_grammar.pl:108:
        PL−Unit: Test adv_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:109:
        PL−Unit: Test adv_2: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:110:
        PL−Unit: Test adv_3: Test succeeded with choicepoint
..
Warning: c:/.../test_grammar.pl:124:
        PL−Unit: Test plural_1: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:126:
        PL−Unit: Test plural_2: Test succeeded with choicepoint
Warning: c:/.../test_grammar.pl:128:
        PL−Unit: Test plural_3: Test succeeded with choicepoint
..
Common supremum found for [alphacell] and [betacell] with [cell]
Warning: c:/.../test_grammar.pl:132:
        PL−Unit: Test plural_6: Test succeeded with choicepoint
 done
% 2 tests failed
% 56 tests passed
false.
```

C.5.2 Test of Reverse Grammar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% test_reverse_grammar.pl:
%
% Helper functions in order to convert parse trees
% to strings (list of words).
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-ensure_loaded([grammar]).
:-ensure_loaded([reverse_grammar]).


% Concerning the unparsing, it is important to check that the successive
% parsings and unparsings are not affecting the tree structure.
% Explicit alignment should then be used when needed.
%
% For instance, the following scenario is to avoid:
%
% ?- np(X,_,[liver,release,of,glucagon,in,blood,by,glycogenolysis,and,by,
% gluconeogenesis],[]), reverse_np(X,Y), np(M,_,Y,[]).
%
% X = np(n(release),
% mod([cn([agent], n(liver)),
% pp(of,[patient], np(n(glucagon), mod([]), ext([]))),
% pp(in,[location], np(n(blood), mod([]), ext([]))),
% pp(by,[manner],np(n(glycogenolysis), mod([]), ext([]))),
% pp(by,[manner],np(n(gluconeogenesis),mod([]),ext([])))]),
% ext([])),
%
% Y = [liver, release, of, glucagon, in, blood, by, glycogenolysis, by,
% gluconeogenesis],
%
% M = np(n(release),
% mod([cn([agent], n(liver)),
% pp(of,[patient], np(n(glucagon), mod([]), ext([]))),
% pp(in,[location], np(n(blood), mod([]), ext([]))),
% pp(by,[manner], np(n(glycogenolysis),
%    mod([pp(by,[manner],np(n(gluconeogenesis),
%        mod([]),
%       ext([])))]),
% ext([])))]),
% ext([])) .
%
% As a consequence, the predicate reverse_np has then been implemented
% in a way that is does not only unparse, but also checks that the
% parsing preserves the meaning.
%
%
test_np(X) :-
            np(T1,Z,X,[]), % Parse noun phrase.
            reverse_np(T1,X2), % Convert it back to string.
            np(T2,Z,X2,[]), !, % Parse new noun phrase.
```

```
        T2 = T1, % Verify that is unchanged.
        reverse_np(T2,X3), % Convert it back to string.
        np(T3,Z,X3,[]), !, % Parse new noun phrase.
        T3 = T1. % Verify that is unchanged.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-begin_tests(reverse_grammar).

% Testing simple case.
test(simple) :-test_np([cell]).

% Testing leaf modifiers.
test(pp) :-test_np([cell,in,pancreas]).
test(rc_active) :-test_np([cell,that,produce,insulin]).
test(rc_passive) :-test_np([insulin,that,is,produced,by,cell]).
test(rc_active_mod) :-test_np([cell,that,produce,in,pancreas,insulin]).
test(rc_passive_mod) :-test_np([insulin,that,is,produced,in,
                                        pancreas,by,cell]).
test(adj) :-test_np([pancreatic,cell]).
test(cn) :-test_np([glucose, production]).
test(ger_1) :-test_np([pancreas,s,production]).
test(ger_2) :-test_np([blood,plasma,s,production]).

% Testing multiple PPs.
test(multiple_1) :-% Explicit alignment.
        test_np([glucose,production,in,pancreas]).
test(multiple_2) :-% Implicit alignment.
        test_np([production,by,liver,of,glucose]).
test(multiple_3) :-% Explicit alignment.
        test_np([production,by,liver,and,of,glucose]).
test(multiple_4) :-% Example of nested structure.
        test_np([production,of,glucose,in,blood]).
test(multiple_5) :-% Example from the comments.
        test_np([liver,release,of,glucagon,in,blood,
                 by,glycogenolysis,and,by,gluconeogenesis]).

% Testing extensions.
test(apposition) :-test_np([insulin,",",a,hormone,","]).
test(apposition) :-test_np([insulin,",",which,isa,hormone,","]).
test(apposition) :-test_np([betacell,",",which,produce,hormone,","]).

:-end_tests(reverse_grammar).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results:

```
?- run_tests.
% PL-Unit: reverse_grammar ...........
ERROR: c:/.../test_reverse_grammar.pl:86:
        test multiple_3: failed


.
ERROR: c:/.../test_reverse_grammar.pl:90:
        test multiple_5: failed


...done
% 2 tests failed
% 16 tests passed
false.
```

C.5.3 Test of Decompose function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% test_decompose.pl:
%
%  Unit tests for extended grammar.
%
%
%
% This file aims at testing the basic features that should be handled by
% Natural Logics grammar.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-ensure_loaded(decompose).
:-[lexicon_insulin].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Helper functions.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% have_same_elements(+X,+Y) − returns true if X and Y are composed of
% the same elements.
%
have_same_elements(X,Y) :-!,
        length(X,L), length(Y,L), !,
        intersection(X,Y,Z), length(Z,L).
intersection([X|Tail],Y,[X|Z]) :-
    member(X,Y),
    intersection(Tail,Y,Z).
intersection([X|Tail],Y,Z) :-
    \+ member(X,Y),
    intersection(Tail,Y,Z).
intersection([],_,[]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Very simple cases.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-begin_tests(decompose).

test(basic_proposition_rel) :-
        decompose([cell, produce, insulin],R),
        R = [fact(prop,[cell], [produce], [insulin], observation)].

test(basic_proposition_isa) :-
        decompose([insulin,isa,hormone],R),
        R = [isa([insulin], [hormone])].
```

```


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of NP modifiers leaf decomposition rules.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

test(basic_pp) :-
        decompose([cell,in,pancreas,produce,insulin],R),
        E = [isa([cell,in,pancreas], [cell]),
             fact(prep, [cell,in,pancreas], in, [pancreas], _),
             fact(prop, [cell,in,pancreas], [produce], [insulin], observation)],
        have_same_elements(R,E).

test(basic_rc) :-
        decompose([cell,that,produce,insulin,produce,insulin],R),
        E = [isa([cell,that,produce,insulin], [cell]),
             fact(rc, [cell,that,produce,insulin], [produce], [insulin], definition),
             fact(prop, [cell,that,produce,insulin], [produce], [insulin], observation)],
        have_same_elements(R,E).

test(basic_adj_intersective) :-
        decompose([pancreatic,cell,produce,insulin],R),
        E = [isa([pancreatic, cell], [cell]),
             isa([pancreatic, cell], [pancreatic, entity]),
             fact(prop, [pancreatic,cell], [produce], [insulin], observation)],
        have_same_elements(R,E).

test(basic_adj_subsective) :-
        decompose([large,cell,produce,insulin],R),
        E = [isa([large, cell], [cell]),
             fact(prop, [large,cell], [produce], [insulin], observation)],
        have_same_elements(R,E).

test(basic_adj_non_subs) :-
        decompose([former,cell,produce,insulin],R),
        R = [fact(prop, [former,cell], [produce], [insulin], observation)].

test(basic_cn) :-
        decompose([insulin, production, produce, insulin],R),
        E = [isa([insulin, production], [production]),
             fact(cn, [insulin, production], none, [insulin], _),
             fact(prop, [insulin, production], [produce], [insulin], observation)],
        have_same_elements(E,R).

test(basic_ger_simple) :-
        decompose([pancreas,s,production, produce, insulin],R),
        E = [isa([pancreas, s, production], [production]),
             fact(ger, [pancreas, s, production], none, [pancreas], _),
             fact(prop, [pancreas, s, production], [produce], [insulin], observation)],
        have_same_elements(E,R).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of NP modifiers general decomposition rules.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

test(basic_cn_multiple) :-
        decompose([insulin, pancreas, production, produce, insulin],R),
        E = [isa([insulin, pancreas, production], [insulin, production]),
             isa([insulin, pancreas, production], [pancreas, production]),
             isa([pancreas, production], [production]),
             isa([insulin, production], [production]),
             fact(cn, [insulin, production], none, [insulin], _),
             fact(cn, [pancreas, production], none, [pancreas], _),
             fact(prop, [insulin, pancreas, production],
                  [produce], [insulin], observation)],
        have_same_elements(E,R).

test(basic_modifiers_multiple) :-
        decompose([large,betacell,in,pancreas, produce, insulin],R),
        E = [isa([large, betacell, in, pancreas], [large, betacell]),
             isa([large, betacell, in, pancreas], [betacell, in, pancreas]),
             isa([large, betacell], [betacell]),
             isa([betacell, in, pancreas], [betacell]),
             fact(prep, [betacell, in, pancreas], in, [pancreas], _),
             fact(prop, [large, betacell, in, pancreas],
                  [produce], [insulin], observation)],
        have_same_elements(E,R).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of NP extension rules.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% One AC,

% One PC - relation


% One PC - isa.
test(extension_pc_isa_simple) :-% Should remove the extension when treated.
        decompose([pancreas, produce, insulin,",",which,isa,hormone,","],R),
        E = [fact(prop, [pancreas], [produce], [insulin], observation),
             isa([insulin], [hormone])],
        have_same_elements(E,R).


test(extension_pc_isa_in_rc) :-% Should remove the extension when treated.
        decompose([pancreas,that,produce,fat,",",which,isa,triglyceride,",",
                   regulate,production],R),
        E = [fact(prop, [pancreas, that, produce, fat], [regulate],
                  [production], observation),
             isa([pancreas, that, produce, fat], [pancreas]),
             fact(rc, [pancreas, that, produce, fat], [produce], [fat], definition),
             isa([fat], [triglyceride])],
        have_same_elements(E,R).

% One PC − isa.
test(extension_pc_isa_in_rc_modified) :-% Should remove the extension when treated.
        decompose([pancreas,that,produce,in,liver,fat,",",which,isa,triglyceride,",",
                    regulate,production],R),
        E = [fact(prop, [pancreas, that, produce, in, liver, fat],
                [regulate], [production], observation),
                isa([pancreas, that, produce, in, liver, fat], [pancreas]),
                fact(rc, [pancreas], [produce, in, liver], [fat], definition),

                attach(fact(rc, [pancreas], [produce, in, liver], [fat], definition),
                        [production, of, fat, by, pancreas, in, liver]),
                isa([fat], [triglyceride]),
                fact(prep, [production, of, fat], of, [fat], result),
                isa([production, of, fat], [production]),
                fact(prep, [production, by, pancreas], by, [pancreas], agent),
                isa([production, by, pancreas], [production]),
                fact(prep, [production, in, liver], in, [liver], location),
                isa([production, in, liver], [production]),
                isa([production, of, fat, by, pancreas, in, liver],
                        [production, by, pancreas, in, liver]),
                isa([production, of, fat, by, pancreas, in, liver],
                        [production, of, fat, by, pancreas]),
                isa([production, of, fat, by, pancreas, in, liver],
                        [production, of, fat, in, liver]),
                isa([production, by, pancreas, in, liver], [production, by, pancreas]),
                isa([production, by, pancreas, in, liver], [production, in, liver]),
                isa([production, of, fat, by, pancreas], [production, by, pancreas]),
                isa([production, of, fat, by, pancreas], [production, of, fat]),
                isa([production, of, fat, in, liver], [production, in, liver]),
                isa([production, of, fat, in, liver], [production, of, fat])],
        have_same_elements(E,R).


% Do also failing cases because not implemented.

% Add case where it is nested far.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Respect of adverbials PPs.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Not implemented yet.

:-end_tests(decompose).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results:

```
 ?- run_tests.
% PL-Unit: decompose
1 relations found.

Warning: c:/.../test_decompose.pl:49:
        PL-Unit: Test basic_proposition_rel: Test succeeded with choicepoint

1 relations found.

Warning: c:/.../test_decompose.pl:53:
        PL-Unit: Test basic_proposition_isa: Test succeeded with choicepoint

3 relations found.

Warning: c:/.../test_decompose.pl:64:
        PL-Unit: Test basic_pp: Test succeeded with choicepoint

3 relations found.

Warning: c:/.../test_decompose.pl:71:
        PL-Unit: Test basic_rc: Test succeeded with choicepoint

3 relations found.

Warning: c:/.../test_decompose.pl:78:
        PL-Unit: Test basic_adj_intersective: Test succeeded with choicepoint

2 relations found.

Warning: c:/.../test_decompose.pl:85:
        PL-Unit: Test basic_adj_subsective: Test succeeded with choicepoint

1 relations found.

Warning: c:/.../test_decompose.pl:91:
        PL-Unit: Test basic_adj_non_subs: Test succeeded with choicepoint

3 relations found.

Warning: c:/.../test_decompose.pl:95:
        PL-Unit: Test basic_cn: Test succeeded with choicepoint

3 relations found.

Warning: c:/.../test_decompose.pl:102:
        PL-Unit: Test basic_ger_simple: Test succeeded with choicepoint

7 relations found.

Warning: c:/.../test_decompose.pl:116:
        PL-Unit: Test basic_cn_multiple: Test succeeded with choicepoint

6 relations found.

Warning: c:/.../test_decompose.pl:128:
        PL-Unit: Test basic_modifiers_multiple: Test succeeded with choicepoint
```

```

2 relations found.

Warning: c:/.../test_decompose.pl:152:
         PL−Unit: Test extension_pc_isa_simple: Test succeeded with choicepoint

4 relations found.

Warning: c:/.../test_decompose.pl:159:
         PL−Unit: Test extension_pc_isa_in_rc: Test succeeded with choicepoint

20 relations found.


Warning: c:/.../test_decompose.pl:170:
         PL−Unit: Test extension_pc_isa_in_rc_modified: Test succeeded with choicepoint
 done
% All 14 tests passed
true.
```

APPENDIX D
Results: application on the insulin article

D.1 Inputs: Sentences & Lexicon

D.1.1 Insulin article

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Insulin_article.pl:
%
% Examples from the two first paragraphs of wikipedia"s
% article on insulin.
%
%
%
% This file contains Natural Logics propositions taken from the two
% first paragraphs of the wikipedia article on insulin (version of
% the 16/11/2016). Each proposition example has an ID composed of
% three digits. The first one refers to the paragraph (1 or 2). The
% second refers to the index of the sentence in the original paragraph.
% The third one is used to distinguished sub-propositions when the
% original sentence has been split in simpler propositions.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% First paragraph.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sentence(110, [insulin, isa, peptide, hormone, that, is, produced, by, betacell,
               of, pancreatic, islet]).

sentence(120, [insulin,
               regulate, by, promotion, of, absorption, of, glucose, from, blood,
               into, fat, into, liver, and, into, skeletal, muscle,
               metabolism, of, carbohydrate]).

sentence(121, [insulin,
               regulate, by, promotion, of, absorption, of, glucose, from, blood,
               into, fat, into, liver, and, into, skeletal, muscle,
               metabolism, of, fat]).
```

```

sentence(122, [insulin,
               regulate, by, promotion, of, absorption, of, glucose, from, blood,
               into, fat, into, liver, and, into, skeletal, muscle,
               metabolism, of, protein]).

sentence(130, [absorbed, glucose,
               is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle,
               cell, into, glycogen, by, glycogenesis]).

sentence(131, [absorbed, glucose,
               is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle,
               cell, into, fat, ",", which, isa, triglyceride, ",", by, lipogenesis]).

sentence(132, [absorbed, glucose,
               is, converted, in, liver, into, glycogen, and, into, fat]).

sentence(140, [glucose, production, by, liver,
               is, inhibited, by,
               high, insulin, concentration, in, blood]).

sentence(141, [glucose, excretion, into, blood, by, liver,
               is, inhibited, by,
               high, insulin, concentration, in, blood]).

sentence(150, [circulating, insulin,
               affect,
               synthesis, in, tissue, of, protein]).

sentence(160, [insulin,
               isa, at, high, insulin, concentration, in, blood,
               anabolic, hormone, that, promote, conversion, of, small, molecule,
               in, blood, into, large, molecule, in, cell]).

sentence(170, [insulin,
               promote, at, low, insulin, concentration,
               widespread, catabolism]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Second paragraph.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sentence(210, [pancreatic, betacell, is, affected, by, glucose, concentration, in,
                blood]).

sentence(220, [betacell, secrete, at, high, glucose, concentration, into, blood,
                insulin]).

sentence(221, [betacell, stop, at, low, glucose, concentration, insulin, secretion,
                into, general, circulation]).

sentence(230, [alphacell, take, from, betacell, cue]).


sentence(231, [alphacell, secrete, into, blood, and, in, opposite, manner, as, betacell,
                    glucagon]).

sentence(232, [alphacell, secrete, highly, at, high, glucose, concentration, glucagon]).

sentence(233, [alphacell, secrete, lowly, at, low, glucose, concentration, glucagon]).

sentence(240, [high, glucagon, concentration, in, blood, plasma, stimulate, powerfully,
                    liver, release, of, glucagon, in, blood, by, glycogenolysis, and, by,
                    gluconeogenesis]).

sentence(241, [stimulation, of, liver, glucose, release, in, blood, by,
                    glycogenolysis, and, by, gluconeogenesis, affect, in, opposite, manner,
                    as, effect, that, is, produced, by, high, insulin, concentration,
                    blood, glucose, concentration]).

sentence(250, [secretion, of, insulin, into, blood, in_response_to, blood, glucose,
                    concentration, isa, primary, mechanism, that, keep, within,
                    narrow, limit, glucose, concentration, in, extracellular, fluid]).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D.1.2 Insulin lexicon

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% lexicon_insulin.pl:
%
% Dedicated lexicon for the first paragraph of
%  the wikipedia article on insulin.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-discontiguous lex/2.
:-discontiguous lex/5.
:-discontiguous nomi/2.


% Lexicon: prepositions
lex(as, preposition).
lex(at, preposition).
lex(by,  preposition).
lex(from, preposition).
lex(in,  preposition).
lex(in_response_to, preposition).
lex(into, preposition).
lex(of, preposition).
lex(within, preposition).


% Lexicon: nouns
lex(absorption, noun, process).
lex(alphacell, noun, cell).
lex(betacell, noun, cell).
lex(blood, noun, body_fluid).
lex(carbohydrate, noun, body_fluid).
lex(catabolism, noun, process).
lex(cell, noun, cell).
lex(circulation, noun, material).
lex(concentration, noun, quantity).
lex(conversion, noun, transformation).
lex(cue, noun, information).
lex(effect, noun, process).
lex(excretion, noun, creation).
lex(fat, noun, body_fluid).
lex(fluid, noun, body_fluid).
lex(glucagon, noun, body_fluid).
lex(gluconeogenesis, noun, creation).
lex(glucose, noun, body_fluid).
lex(glycogen, noun, body_fluid).
lex(glycogenesis, noun, creation).
lex(glycogenolysis, noun, creation).
lex(hormone, noun, protein).
lex(impact, noun, enhancement).
lex(insulin, noun, protein).
lex(islet, noun, body_part).
lex(keeping, noun, process).
lex(limit, noun, quantity).
lex(lipogenesis, noun, creation).
lex(liver, noun, organ).
lex(manner, noun, quality).
lex(mechanism, noun, process).
lex(metabolism, noun, process).
lex(molecule, noun, mass).
lex(muscle, noun, body_part).
lex(pancreas, noun, organ).
lex(peptide, noun, protein).
lex(plasma, noun, material).
lex(promotion, noun, enhancement).
lex(production, noun, creation).
lex(protein, noun, protein).
lex(regulation, noun, enhancement).
lex(release, noun, process).
lex(secretion, noun, creation).
lex(stimulation, noun, enhancement).
lex(stop, noun, enhancement).
lex(synthesis, noun, creation).
lex(taking, noun, process).
lex(tissue, noun, body_part).
lex(triglyceride, noun, body_fluid).


% Lexicon: verbs
lex(isa, trans, copular, is_some, const(X,X)).

lex(affect, trans, event, affected, const([_],[_])).
lex(absorb, trans, event, absorbed, const([body_part, process],
                                                       [body_fluid])).
lex(convert, trans, event, converted, const([body_part, transformation],
                                                       [body_fluid])).
lex(excrete, trans, event, excreted, const([creative, body_part],
                                                       [body_fluid])).
lex(inhibit, trans, event, inhibited, const([process, quality,
                                                        body_part, body_fluid],
                                                       [process])).
lex(keep, trans, event, kept, const([process, quality,
                                                        body_part, body_fluid],
                                                       [process, quality,
                                                        body_part, body_fluid])).
lex(produce, trans, event, produced, const([body_part, creation],
                                                       [body_fluid])).
lex(produce, trans, event, produced, const([quality], [entity])). % Abstract sense
lex(promote, trans, event, promoted, const([process, quality,
                                                        body_part, body_fluid],
                                                       [process])).
lex(release, trans, event, released, const([body_part, process],
                                                       [body_fluid])).
lex(regulate, trans, event, regulated, const([process, quality,
                                                        body_part, body_fluid],
                                                       [process, state])).
lex(secrete, trans, event, secreted, const([body_part, creation],
                                                       [body_fluid])).
lex(stimulate, trans, event, stimulated, const([process, quality,
                                                        body_part, body_fluid],
                                                    [process, state])).
lex(stop, trans, event, stopped, const([process, body_part],
                                                    [process, state])).
lex(synthesize, trans, event, synthesized, const([process],
                                                    [body_fluid])).
lex(take, trans, event, taken, const(_,_)).

% Equivalence verbs – nouns.
nomi(_, isa).
nomi(absorption, absorb).
nomi(excretion, excrete).
nomi(conversion, convert).
nomi(impact, affect).
nomi(keeping, keep).
nomi(production, produce).
nomi(promotion, promote).
nomi(regulation, regulate).
nomi(release, release).
nomi(secretion, secrete).
nomi(stimulation, stimulate).
nomi(stop, stop).
nomi(synthesis, synthesize).
nomi(taking, take).


% Lexicon: adjectives:
lex(absorbed, adj, inter, predi, const([body_fluid])).
lex(anabolic, adj, inter, predi, const([body_fluid, body_part])).
lex(extracellular, adj, inter, predi, const([body_fluid, body_part])).
lex(circulating, adj, inter, predi, const([_])).
lex(pancreatic, adj, inter, predi, const([body_fluid, body_part])).
lex(skeletal, adj, inter, predi, const([body_fluid, body_part])).

lex(general, adj, subs, predi, const([_])).
lex(high, adj, subs, predi, const([_])).
lex(large, adj, subs, predi, const([_])).
lex(low, adj, subs, predi, const([_])).
lex(narrow, adj, subs, predi, const([_])).
lex(powerful, adj, subs, predi, const([_])).
lex(small, adj, subs, predi, const([_])).
lex(strong, adj, subs, predi, const([_])).
lex(widespread, adj, subs, predi, const([_])).
lex(primary, adj, subs, non_predi, const([_])).

lex(opposite, adj, non_subs, predi, const([process, quality])).
lex(former, adj, non_subs, predi, const([_])).


% Lexicon: adverbs
lex(highly, adv).
lex(lowly, adv).
lex(powerfully, adv).
lex(strongly, adv).

% Equivalence adjectives – adverbs.
adv_adj(highly, high).
adv_adj(lowly, low).
adv_adj(powerfully, powerful).
adv_adj(strongly, strong).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D.2 Automated tests on the insulin article

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% test_insulin_article.pl:
%
% Test algorithms on the first paragraph of wikipedia"s
% article on insulin.
%
%
%
% This file aims at defining test cases for Natural Logics parser,
% proposition decomposition and graph search using ontological analysis
% of concepts.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:-ensure_loaded(insulin_article). % Loading sentences to test.
:-ensure_loaded(lexicon_insulin). % Loading the lexicon.
:-ensure_loaded(grammar). % Loading the grammar.
:-ensure_loaded(decompose). % Loading decomposition algo.
:-ensure_loaded(infer). % Loading inference rules.
:-ensure_loaded(subsum). % Loading subsumption rules.
:-ensure_loaded(search). % Loading search algorithm.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Definition of functions to create the knowledge base.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

create_kb(Title) :-
        read_article(R),
        infer(R,R1),
        % subsum(R1,R2),
        save_as_text(R1,Title).

% save_as_text(+KB, +Title) - save the KB as a text file.
save_as_text(KB, Title) :-
    open(Title,write, Stream), write_kb(Stream,KB), close(Stream).
write_kb(_,[]).
write_kb(Stream,[H|T]) :-write(Stream,H), write(Stream,"."), nl(Stream),
        write_kb(Stream,T).

% read_from_text(+Title, -KB) - get the KB from text file.
read_from_text(Title, KB):-
    open(Title, read, Str), read_file(Str,X), lbl(X,KB), close(Str).
read_file(Stream,[]) :-at_end_of_stream(Stream).
read_file(Stream,[X|L]) :-\+ at_end_of_stream(Stream),
    read(Stream,X), read_file(Stream,L).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% Definition of some helper functions.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

decompose_sentence(X,R) :-sentence(X,Y), decompose(Y,R).
is_sentence(X) :-sentence(X, _).

read_article(R) :-setof(Y,is_sentence(Y),IDs), read_article(IDs,X), filter(X,R).
read_article([],[]).
read_article([H|T], R) :-read_article(T, R0),
            decompose_sentence(H,R1), append(R0,R1,R).

% lbl(+X,-Y) :-returns list except last element.
lbl([X|Xs], Ys) :-list_butlast_prev(Xs, Ys, X).
list_butlast_prev([], [], _).
list_butlast_prev([X1|Xs], [X0|Ys], X0) :-
    list_butlast_prev(Xs, Ys, X1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Definition of basic test cases.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-begin_tests(insulin).

% Parsing of the sentences, should be successful.
test(sentence_110) :-sentence(110,X), p(_,X,[]).
test(sentence_120) :-sentence(120,X), p(_,X,[]).
test(sentence_121) :-sentence(121,X), p(_,X,[]).
test(sentence_122) :-sentence(122,X), p(_,X,[]).
test(sentence_130) :-sentence(130,X), p(_,X,[]).
test(sentence_131) :-sentence(131,X), p(_,X,[]).
test(sentence_132) :-sentence(132,X), p(_,X,[]).
test(sentence_140) :-sentence(140,X), p(_,X,[]).
test(sentence_141) :-sentence(141,X), p(_,X,[]).
test(sentence_150) :-sentence(150,X), p(_,X,[]).
test(sentence_160) :-sentence(160,X), p(_,X,[]).
test(sentence_171) :-sentence(170,X), p(_,X,[]).

test(sentence_210) :-sentence(210,X), p(_,X,[]).
test(sentence_220) :-sentence(220,X), p(_,X,[]).
test(sentence_221) :-sentence(221,X), p(_,X,[]).
test(sentence_230) :-sentence(230,X), p(_,X,[]).
test(sentence_231) :-sentence(231,X), p(_,X,[]).
test(sentence_232) :-sentence(232,X), p(_,X,[]).
test(sentence_233) :-sentence(233,X), p(_,X,[]).
test(sentence_240) :-sentence(240,X), p(_,X,[]).
test(sentence_241) :-sentence(241,X), p(_,X,[]).
test(sentence_250) :-sentence(250,X), p(_,X,[]).

% Decomposition of the sentences, should be successful (considering
% existing errors in the parse tree).
test(decompose_110) :-sentence(110,X), !, decompose(X,_).
test(decompose_120) :-sentence(120,X), !, decompose(X,_).
test(decompose_121) :-sentence(121,X), !, decompose(X,_).
test(decompose_122) :-sentence(122,X), !, decompose(X,_).
test(decompose_130) :-sentence(130,X), !, decompose(X,_).
test(decompose_131) :-sentence(131,X), !, decompose(X,_).
test(decompose_132) :-sentence(132,X), !, decompose(X,_).
test(decompose_140) :-sentence(140,X), !, decompose(X,_).
test(decompose_141) :-sentence(141,X), !, decompose(X,_).
test(decompose_150) :-sentence(150,X), !, decompose(X,_).
test(decompose_160) :-sentence(160,X), !, decompose(X,_).
test(decompose_170) :-sentence(170,X), !, decompose(X,_).

test(decompose_210) :-sentence(210,X), !, decompose(X,_).
test(decompose_220) :-sentence(220,X), !, decompose(X,_).
test(decompose_221) :-sentence(221,X), !, decompose(X,_).
test(decompose_230) :-sentence(230,X), !, decompose(X,_).
test(decompose_231) :-sentence(231,X), !, decompose(X,_).
test(decompose_232) :-sentence(232,X), !, decompose(X,_).
test(decompose_233) :-sentence(233,X), !, decompose(X,_).
test(decompose_240) :-sentence(240,X), !, decompose(X,_).
test(decompose_241) :-sentence(241,X), !, decompose(X,_).
test(decompose_250) :-sentence(250,X), !, decompose(X,_).

:-end_tests(insulin).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results:

```
?- run_tests.
% PL-Unit: insulin
Warning: [Thread pce] c:/.../test_insulin_article.pl:84:
         PL-Unit: Test sentence_110: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:85:
         PL-Unit: Test sentence_120: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:86:
         PL-Unit: Test sentence_121: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:87:
         PL-Unit: Test sentence_122: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:88:
         PL-Unit: Test sentence_130: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:89:
         PL-Unit: Test sentence_131: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:90:
         PL-Unit: Test sentence_132: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:91:
         PL-Unit: Test sentence_140: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:92:
         PL-Unit: Test sentence_141: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:93:
         PL-Unit: Test sentence_150: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:94:
         PL-Unit: Test sentence_160: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:95:
         PL-Unit: Test sentence_171: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:97:
         PL-Unit: Test sentence_210: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:98:
         PL-Unit: Test sentence_220: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:99:
         PL-Unit: Test sentence_221: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:100:
         PL-Unit: Test sentence_230: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:101:
         PL-Unit: Test sentence_231: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:102:
         PL-Unit: Test sentence_232: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:103:
         PL-Unit: Test sentence_233: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:104:
         PL-Unit: Test sentence_240: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:105:
         PL-Unit: Test sentence_241: Test succeeded with choicepoint
Warning: [Thread pce] c:/.../test_insulin_article.pl:106:
         PL-Unit: Test sentence_250: Test succeeded with choicepoint

11 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:110:
         PL-Unit: Test decompose_110: Test succeeded with choicepoint

108 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:111:
         PL-Unit: Test decompose_120: Test succeeded with choicepoint
```

```

108 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:112:
          PL-Unit: Test decompose_121: Test succeeded with choicepoint

108 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:113:
          PL-Unit: Test decompose_122: Test succeeded with choicepoint

208 relations found.

.
210 relations found.

.
40 relations found.

.
21 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:117:
          PL-Unit: Test decompose_140: Test succeeded with choicepoint

30 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:118:
          PL-Unit: Test decompose_141: Test succeeded with choicepoint

7 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:119:
          PL-Unit: Test decompose_150: Test succeeded with choicepoint

Default observation.

27 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:120:
          PL-Unit: Test decompose_160: Test succeeded with choicepoint

23 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:121:
          PL-Unit: Test decompose_170: Test succeeded with choicepoint

9 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:123:
          PL-Unit: Test decompose_210: Test succeeded with choicepoint

43 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:124:
          PL-Unit: Test decompose_220: Test succeeded with choicepoint

```

```
29 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:125:
        PL−Unit: Test decompose_221: Test succeeded with choicepoint

17 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:126:
        PL−Unit: Test decompose_230: Test succeeded with choicepoint

42 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:127:

 PL−Unit: Test decompose_231: Test succeeded with choicepoint

42 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:128:
        PL−Unit: Test decompose_232: Test succeeded with choicepoint

42 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:129:
        PL−Unit: Test decompose_233: Test succeeded with choicepoint

117 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:130:
        PL−Unit: Test decompose_240: Test succeeded with choicepoint

121 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:131:
        PL−Unit: Test decompose_241: Test succeeded with choicepoint

50 relations found.

Warning: [Thread pce] c:/.../test_insulin_article.pl:132:
        PL−Unit: Test decompose_250: Test succeeded with choicepoint
 done
% All 44 tests passed
true.
```

D.3 Parsing of the article sentences (without ontology)

```
% First paragraph
?− sentence(110,X), p(Z,X,[]).
X = [insulin, isa, peptide, hormone, that, is, produced, by, betacell, of, pancreatic, islet],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(isa), mod([])), np(n(hormone),
     mod([cn(n(peptide)), rc(verb(passive(produced), mod([])), np(n(betacell), mod([pp(of,
     np(n(islet), mod([adj(pancreatic)]), ext([])))]), ext([])))]), ext([]))))] .

?− sentence(120,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into, fat, into,
     liver, and, into, skeletal, muscle, metabolism, of, carbohydrate],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, np(n(promotion),
     mod([pp(of, np(n(absorption), mod([pp(of, np(n(glucose), mod([pp(from, np(n(blood),
     mod([pp(into, np(n(fat), mod([pp(into, np(n(liver), mod([]), ext([]))), pp(into, np(n(muscle),
     mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))]), ext([])))]), ext([])))]),
     ext([])))])), np(n(metabolism), mod([pp(of, np(n(carbohydrate), mod([]), ext([])))]),
     ext([]))))] .

?− sentence(121,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into, fat, into,
     liver, and, into, skeletal, muscle, metabolism, of, fat],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, np(n(promotion),
     mod([pp(of, np(n(absorption), mod([pp(of, np(n(glucose), mod([pp(from, np(n(blood),
     mod([pp(into, np(n(fat), mod([pp(into, np(n(liver), mod([]), ext([]))), pp(into, np(n(muscle),
     mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))]), ext([])))]), ext([])))]),
     ext([])))])), np(n(metabolism), mod([pp(of, np(n(fat), mod([]), ext([])))]), ext([]))))] .

?− sentence(122,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into, fat, into,
     liver, and, into, skeletal, muscle, metabolism, of, protein],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, np(n(promotion),
     mod([pp(of, np(n(absorption), mod([pp(of, np(n(glucose), mod([pp(from, np(n(blood),
     mod([pp(into, np(n(fat), mod([pp(into, np(n(liver), mod([]), ext([]))), pp(into, np(n(muscle),
     mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))]), ext([])))]), ext([])))]),
     ext([])))])), np(n(metabolism), mod([pp(of, np(n(protein), mod([]), ext([])))]), ext([]))))] .

?− sentence(130,X),p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle, cell,
     into, glycogen, by, glycogenesis],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
     np(n(fat), mod([]), ext([]))), pp(in, np(n(liver), mod([]), ext([]))), pp(in, np(n(cell),
     mod([adj(skeletal), cn(n(muscle)), pp(into, np(n(glycogen), mod([]), ext([])))]), ext([])))])),
     np(n(glycogenesis), mod([]), ext([]))))] .

?− sentence(131,X),p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle, cell,
     into, fat, ",", which, isa, triglyceride, ",", by, lipogenesis],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
     np(n(fat), mod([]), ext([]))), pp(in, np(n(liver), mod([]), ext([]))), pp(in, np(n(cell),
     mod([adj(skeletal), cn(n(muscle)), pp(into, np(n(fat), mod([]), ext([pc(verb(active(isa),
     mod([])), np(n(triglyceride), mod([]), ext([])))])))]), ext([])))])), np(n(lipogenesis),
     mod([]), ext([]))))] .

?− sentence(132,X),p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, liver, into, glycogen, and, into, fat],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
    np(n(liver), mod([pp(into, np(n(glycogen), mod([]), ext([]))), pp(into, np(n(fat), mod([]),
    ext([])))]), ext([]))])))))] .

?− sentence(140,X), p(Z,X,[]).
X = [glucose, production, by, liver, is, inhibited, by, high, insulin, concentration, in, blood],
Z = [p(np(n(production), mod([cn(n(glucose)), pp(by, np(n(liver), mod([]), ext([])))]), ext([])),
    vp(verb(passive(inhibited), mod([])), np(n(concentration), mod([adj(high), cn(n(insulin)),
    pp(in, np(n(blood), mod([]), ext([])))]), ext([]))))]

?− sentence(141,X), p(Z,X,[]).
X = [glucose, excretion, into, blood, by, liver, is, inhibited, by, high, insulin, concentration, in,
    blood],
Z = [p(np(n(excretion), mod([cn(n(glucose)), pp(into, np(n(blood), mod([pp(by, np(n(liver),
    mod([]), ext([])))]), ext([])))]), ext([])), vp(verb(passive(inhibited), mod([])),
    np(n(concentration), mod([adj(high), cn(n(insulin)), pp(in, np(n(blood), mod([]), ext([])))]),
    ext([]))))] .

?− sentence(150,X), p(Z,X,[]).
X = [circulating, insulin, affect, synthesis, in, tissue, of, protein],
Z = [p(np(n(insulin), mod([adj(circulating)]), ext([])), vp(verb(active(affect), mod([])),
    np(n(synthesis), mod([pp(in, np(n(tissue), mod([pp(of, np(n(protein), mod([]), ext([])))]),
    ext([])))]), ext([]))))]

?− sentence(160,X), p(Z,X,[]).
X = [insulin, isa, at, high, insulin, concentration, in, blood, anabolic, hormone, that, promote,
    conversion, of, small, molecule, in, blood, into, large, molecule, in, cell],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(isa), mod([pp(at, np(n(concentration),
    mod([adj(high), cn(n(insulin)), pp(in, np(n(blood), mod([]), ext([])))]), ext([])))])),
    np(n(hormone), mod([adj(anabolic), rc(verb(active(promote), mod([])), np(n(conversion),
    mod([pp(of, np(n(molecule), mod([adj(small), pp(in, np(n(blood), mod([pp(into,
    np(n(molecule), mod([adj(large), pp(in, np(n(cell), mod([]), ext([])))]), ext([])))]),
    ext([])))]), ext([])))]), ext([])))]), ext([]))))] .

?− sentence(170,X), p(Z,X,[]).
X = [insulin, promote, at, low, insulin, concentration, widespread, catabolism],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(promote), mod([pp(at, np(n(concentration),
    mod([adj(low), cn(n(insulin))]), ext([])))])), np(n(catabolism), mod([adj(widespread)]),
    ext([]))))]

% Second paragraph
?− sentence(210,X), p(Z,X,[]).
X = [pancreatic, betacell, is, affected, by, glucose, concentration, in, blood],
Z = [p(np(n(betacell), mod([adj(pancreatic)]), ext([])), vp(verb(passive(affected), mod([])),
    np(n(concentration), mod([cn(n(glucose)), pp(in, np(n(blood), mod([]), ext([])))]), ext([]))))] .

?− sentence(220,X), p(Z,X,[]).
X = [betacell, secrete, at, high, glucose, concentration, insulin],
Z = [p(np(n(betacell), mod([]), ext([])), vp(verb(active(secrete), mod([pp(at, np(n(glucose),
    mod([adj(high)]), ext([])))])), np(n(insulin), mod([cn(n(concentration))]), ext([]))))] .

?− sentence(221,X), p(Z,X,[]).
X = [betacell, stop, at, low, glucose, concentration, insulin, secretion, into, general, circulation],
Z = [p(np(n(betacell), mod([]), ext([])), vp(verb(active(stop), mod([pp(at, np(n(glucose),
    mod([adj(low)]), ext([])))])), np(n(secretion), mod([cn(n(concentration)), cn(n(insulin)),
    pp(into, np(n(circulation), mod([adj(general)]), ext([])))]), ext([]))))] .

?− sentence(230,X), p(Z,X,[]).
```

```
X = [alphacell, take, from, betacell, cue],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(take), mod([pp(from, np(n(betacell),
    mod([]), ext([]))))])), np(n(cue), mod([]), ext([]))))] .

?- sentence(231,X), p(Z,X,[]).
X = [alphacell, secrete, into, blood, and, in, opposite, manner, as, betacell, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([pp(into, np(n(blood),
    mod([]), ext([]))), pp(in, np(n(manner), mod([adj(opposite), pp(as, np(n(betacell), mod([]),
    ext([]))))]), ext([]))))])), np(n(glucagon), mod([]), ext([]))))]

?- sentence(232,X), p(Z,X,[]).
X = [alphacell, secrete, highly, at, high, glucose, concentration, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([highly, pp(at,
    np(n(glucose), mod([adj(high)]), ext([]))))])), np(n(glucagon), mod([cn(n(concentration))]),
    ext([]))))] .

?- sentence(233,X), p(Z,X,[]).
X = [alphacell, secrete, lowly, at, low, glucose, concentration, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([lowly, pp(at,
    np(n(glucose), mod([adj(low)]), ext([]))))])), np(n(glucagon), mod([cn(n(concentration))]),
    ext([]))))] .

?- sentence(240,X), p(Z,X,[]).
X = [high, glucagon, concentration, in, blood, plasma, stimulate, powerfully, liver, release, of,
    glucagon, in, blood, by, glycogenolysis, and, by, gluconeogenesis],
Z = [p(np(n(concentration), mod([adj(high), cn(n(glucagon)), pp(in, np(n(plasma),
    mod([cn(n(blood))]), ext([])))]), ext([])), vp(verb(active(stimulate), mod([powerfully])),
    np(n(release), mod([cn(n(liver)), pp(of, np(n(glucagon), mod([pp(in, np(n(blood), mod([pp(by,
    np(n(glycogenolysis), mod([]), ext([]))), pp(by, np(n(gluconeogenesis), mod([]), ext([])))]),
    ext([])))]), ext([])))]), ext([]))))]

?- sentence(241,X), p(Z,X,[]).
X = [stimulation, of, liver, glucose, release, in, blood, by, glycogenolysis, and, by,
    gluconeogenesis, affect, in, opposite, manner, as, effect, that, is, produced, by, high,
    insulin, concentration, blood, glucose, concentration],
Z = [p(np(n(stimulation), mod([pp(of, np(n(release), mod([cn(n(liver)), cn(n(glucose)), pp(in,
    np(n(blood), mod([pp(by, np(n(glycogenolysis), mod([]), ext([]))), pp(by,
    np(n(gluconeogenesis), mod([]), ext([])))]), ext([])))]), ext([])))]), ext([])),
    vp(verb(active(affect), mod([pp(in, np(n(manner), mod([adj(opposite), pp(as, np(n(effect),
    mod([rc(verb(passive(produced), mod([])), np(n(insulin), mod([adj(high)]), ext([])))]),
    ext([])))]), ext([])))])), np(n(concentration), mod([cn(n(concentration)), cn(n(blood)),
    cn(n(glucose))]), ext([]))))] .

?- sentence(250,X), p(Z,X,[]).
X = [secretion, of, insulin, into, blood, in_response_to, blood, glucose, concentration, isa,
    primary, mechanism, that, keep, within, narrow, limit, glucose, concentration, in,
    extracellular, fluid],
Z = [p(np(n(secretion), mod([pp(of, np(n(insulin), mod([pp(into, np(n(blood),
    mod([pp(in_response_to, np(n(concentration), mod([cn(n(blood)), cn(n(glucose))]), ext([])))]),
    ext([])))]), ext([])))]), ext([])), vp(verb(active(isa), mod([])), np(n(mechanism),
    mod([adj(primary), rc(verb(active(keep), mod([pp(within, np(n(limit), mod([adj(narrow)]),
    ext([])))])), np(n(concentration), mod([cn(n(glucose)), pp(in, np(n(fluid),
    mod([adj(extracellular)]), ext([])))]), ext([])))]), ext([]))))] .
```

D.4 Parsing of the article sentences (with ontology)

```
% First paragraph.
?− sentence(110,X), p(Z,X,[]).
X = [insulin, isa, peptide, hormone, that, is, produced, by, betacell, of, pancreatic, islet],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(isa), mod([])), np(n(hormone),
    mod([cn([characterization], n(peptide)), rc(verb(passive(produced), mod([])),
    np(n(betacell), mod([pp(of, [part_of], np(n(islet), mod([adj(pancreatic)]),
    ext([])))]), ext([])))]), ext([]))))] .

?− sentence(120,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into,
    fat, into, liver, and, into, skeletal, muscle, metabolism, of, carbohydrate],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, [manner],
    np(n(promotion), mod([pp(of, [patient], np(n(absorption), mod([pp(of, [patient],
    np(n(glucose), mod([]), ext([]))), pp(from, [source], np(n(blood), mod([]), ext([]))),
    pp(into, [direction], np(n(fat), mod([]), ext([]))), pp(into, [direction],
    np(n(liver), mod([]), ext([]))), pp(into, [direction], np(n(muscle),
    mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))])), np(n(metabolism),
    mod([pp(of, [patient], np(n(carbohydrate), mod([]), ext([])))]), ext([]))))]

?− sentence(121,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into,
    fat, into, liver, and, into, skeletal, muscle, metabolism, of, fat],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, [manner],
    np(n(promotion), mod([pp(of, [patient], np(n(absorption), mod([pp(of, [patient],
    np(n(glucose), mod([]), ext([]))), pp(from, [source], np(n(blood), mod([]), ext([]))),
    pp(into, [direction], np(n(fat), mod([]), ext([]))), pp(into, [direction],
    np(n(liver), mod([]), ext([]))), pp(into, [direction], np(n(muscle),
    mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))])), np(n(metabolism),
    mod([pp(of, [patient], np(n(fat), mod([]), ext([])))]), ext([]))))]

?− sentence(122,X), p(Z,X,[]).
X = [insulin, regulate, by, promotion, of, absorption, of, glucose, from, blood, into,
    fat, into, liver, and, into, skeletal, muscle, metabolism, of, protein],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(regulate), mod([pp(by, [manner],
    np(n(promotion), mod([pp(of, [patient], np(n(absorption), mod([pp(of, [patient],
    np(n(glucose), mod([]), ext([]))), pp(from, [source], np(n(blood), mod([]), ext([]))),
    pp(into, [direction], np(n(fat), mod([]), ext([]))), pp(into, [direction],
    np(n(liver), mod([]), ext([]))), pp(into, [direction], np(n(muscle),
    mod([adj(skeletal)]), ext([])))]), ext([])))]), ext([])))])), np(n(metabolism),
    mod([pp(of, [patient], np(n(protein), mod([]), ext([])))]), ext([]))))]

\iffalse
?− sentence(130,X), p(Z,X,[]).
X = [glucose, is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle, cell,
    into, glycogen, and, into, fat, ",", which, isa, triglyceride, ","],
Z = [p(np(n(glucose), mod([]), ext([])), vp(verb(passive(converted), mod([pp(in,
    [location], np(n(fat), mod([]), ext([]))), pp(in, [location], np(n(liver), mod([]),
    ext([]))), pp(in, [location], np(n(cell), mod([adj(skeletal), cn([characterization],
    n(muscle))]), ext([]))), pp(into, [result], np(n(glycogen), mod([]), ext([]))),
    pp(into, [result], np(n(fat), mod([]), ext([pc(verb(active(isa), mod([])),
    np(n(triglyceride), mod([]), ext([])))])))])))))] .
\fi
?− sentence(130,X), p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle,
    cell, into, glycogen, by, glycogenesis],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
    [location], np(n(fat), mod([]), ext([]))), pp(in, [location], np(n(liver), mod([]),
    ext([]))), pp(in, [location], np(n(cell), mod([adj(skeletal), cn([characterization],
    n(muscle))]), ext([]))), pp(into, [result], np(n(glycogen), mod([]), ext([]))), pp(by,
    [manner], np(n(glycogenesis), mod([]), ext([])))])))] .

?− sentence(131,X), p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, fat, ",", in, liver, and, in, skeletal, muscle,
    cell, into, fat, ",", which, isa, triglyceride, ",", by, lipogenesis],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
    [location], np(n(fat), mod([]), ext([]))), pp(in, [location], np(n(liver), mod([]),
    ext([]))), pp(in, [location], np(n(cell), mod([adj(skeletal), cn([characterization],
    n(muscle))]), ext([]))), pp(into, [result], np(n(fat), mod([]), ext([pc(verb(active(isa),
    mod([])), np(n(triglyceride), mod([]), ext([])))]))), pp(by, [manner], np(n(lipogenesis),
    mod([]), ext([])))])))]

?− sentence(132,X), p(Z,X,[]).
X = [absorbed, glucose, is, converted, in, liver, into, glycogen, and, into, fat],
Z = [p(np(n(glucose), mod([adj(absorbed)]), ext([])), vp(verb(passive(converted), mod([pp(in,
    [location], np(n(liver), mod([]), ext([]))), pp(into, [result], np(n(glycogen), mod([]),
    ext([]))), pp(into, [result], np(n(fat), mod([]), ext([])))])))] .

?− sentence(140,X), p(Z,X,[]).
X = [glucose, production, by, liver, is, inhibited, by, high, insulin, concentration, in ,blood],
Z = [p(np(n(production), mod([cn([result], n(glucose)), pp(by, [agent], np(n(liver),
    mod([]), ext([])))]), ext([])), vp(verb(passive(inhibited), mod([])),
    np(n(concentration), mod([adj(high), cn([bearer], n(insulin)), pp(in, [location],
    np(n(blood), mod([]), ext([])))]), ext([]))))] .

?− sentence(141,X), p(Z,X,[]).
X = [glucose, excretion, into, blood, by, liver, is, inhibited, by, high, insulin,
    concentration, in, blood],
Z = [p(np(n(excretion), mod([cn([result], n(glucose)), pp(into, [direction], np(n(blood),
    mod([]), ext([]))), pp(by, [agent], np(n(liver), mod([]), ext([])))]), ext([])),
    vp(verb(passive(inhibited), mod([])), np(n(concentration), mod([adj(high),
    cn([bearer], n(insulin)), pp(in, [location], np(n(blood), mod([]), ext([])))]),
    ext([]))))] .

?− sentence(150,X), p(Z,X,[]).
X = [circulating, insulin, affect, synthesis, in, tissue, of, protein],
Z = [p(np(n(insulin), mod([adj(circulating)]), ext([])), vp(verb(active(affect), mod([])),
    np(n(synthesis), mod([pp(in, [location], np(n(tissue), mod([pp(of, [part_of],
    np(n(protein), mod([]), ext([])))]), ext([])))]), ext([]))))] .

?− sentence(160,X), p(Z,X,[]).
X = [insulin, isa, at, high, insulin, concentration, in, blood, anabolic, hormone, that,
    promote, conversion, of, small, molecule, in, blood, into, large, molecule, in, cell],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(isa), mod([pp(at, [condition],
    np(n(concentration), mod([adj(high), cn([bearer], n(insulin)), pp(in, [location],
    np(n(blood), mod([]), ext([])))]), ext([])))])), np(n(hormone), mod([adj(anabolic),
    rc(verb(active(promote), mod([])), np(n(conversion), mod([pp(of, [patient],
    np(n(molecule), mod([adj(small)]), ext([]))), pp(in, [location], np(n(blood), mod([]),
    ext([]))), pp(into, [result], np(n(molecule), mod([adj(large), pp(in, [location],
    np(n(cell), mod([]), ext([])))]), ext([])))]), ext([])))]), ext([]))))] .

?− sentence(170,X), p(Z,X,[]).
X = [insulin, promote, at, low, insulin, concentration, widespread, catabolism],
Z = [p(np(n(insulin), mod([]), ext([])), vp(verb(active(promote), mod([pp(at, [condition],
    np(n(concentration), mod([adj(low), cn([bearer], n(insulin))]), ext([])))])),
    np(n(catabolism), mod([adj(widespread)]), ext([]))))] .


% Second paragraph.
?− sentence(210,X), p(Z,X,[]).
X = [pancreatic, betacell, is, affected, by, glucose, concentration, in, blood],
Z = [p(np(n(betacell), mod([adj(pancreatic)]), ext([])), vp(verb(passive(affected), mod([])),
    np(n(concentration), mod([cn([bearer], n(glucose)), pp(in, [location], np(n(blood), mod([]),
    ext([])))]), ext([]))))]

?− sentence(220,X), p(Z,X,[]).
X = [betacell, secrete, at, high, glucose, concentration, insulin],
Z = [p(np(n(betacell), mod([]), ext([])), vp(verb(active(secrete), mod([pp(at, [condition],
    np(n(concentration), mod([adj(high), cn([bearer], n(glucose))]), ext([])))])), np(n(insulin),
    mod([]), ext([]))))] .

?− sentence(221,X), p(Z,X,[]).
X = [betacell, stop, at, low, glucose, concentration, insulin, secretion, into, general, circulation],
Z = [p(np(n(betacell), mod([]), ext([])), vp(verb(active(stop), mod([pp(at, [condition],
    np(n(concentration), mod([adj(low), cn([bearer], n(glucose))]), ext([])))])), np(n(secretion),
    mod([cn([result], n(insulin)), pp(into, [direction], np(n(circulation), mod([adj(general)]),
    ext([])))]), ext([]))))] .

?− sentence(230,X), p(Z,X,[]).
X = [alphacell, take, from, betacell, cue],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(take), mod([pp(from, [source],
    np(n(betacell), mod([]), ext([])))])), np(n(cue), mod([]), ext([]))))] .

?− sentence(231,X), p(Z,X,[]).
X = [alphacell, secrete, into, blood, and, in, opposite, manner, as, betacell, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([pp(into, [direction],
    np(n(blood), mod([]), ext([]))), pp(in, [manner], np(n(manner), mod([adj(opposite), pp(as,
    [comparison], np(n(betacell), mod([]), ext([])))]), ext([])))])), np(n(glucagon), mod([]),
    ext([]))))] .

?− sentence(232,X), p(Z,X,[]).
X = [alphacell, secrete, highly, at, high, glucose, concentration, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([adv(highly), pp(at,
    [condition], np(n(concentration), mod([adj(high), cn([bearer], n(glucose))]), ext([])))])),
    np(n(glucagon), mod([]), ext([]))))] .

?− sentence(233,X), p(Z,X,[]).
X = [alphacell, secrete, lowly, at, low, glucose, concentration, glucagon],
Z = [p(np(n(alphacell), mod([]), ext([])), vp(verb(active(secrete), mod([adv(lowly), pp(at,
    [condition], np(n(concentration), mod([adj(low), cn([bearer], n(glucose))]), ext([])))])),
    np(n(glucagon), mod([]), ext([]))))]

?− sentence(240,X), p(Z,X,[]).
X = [high, glucagon, concentration, in, blood, plasma, stimulate, powerfully, liver, release, of,
    glucagon, in, blood, by, glycogenolysis, and, by, gluconeogenesis],
Z = [p(np(n(concentration), mod([adj(high), cn([bearer], n(glucagon)), pp(in, [location],
    np(n(plasma), mod([cn([characterization], n(blood))]), ext([])))]), ext([])),
    vp(verb(active(stimulate), mod([adv(powerfully)])), np(n(release), mod([cn([agent], n(liver)),
    pp(of, [patient], np(n(glucagon), mod([]), ext([]))), pp(in, [location], np(n(blood), mod([]),
    ext([]))), pp(by, [manner], np(n(glycogenolysis), mod([]), ext([]))), pp(by, [manner],
    np(n(gluconeogenesis), mod([]), ext([]))))]), ext([]))))]

?− sentence(241,X), p(Z,X,[]).
X = [stimulation, of, liver, glucose, release, in, blood, by, glycogenolysis, and, by,
    gluconeogenesis, affect, in, opposite, manner, as, effect, that, is, produced, by, high,
    insulin, concentration, blood, glucose, concentration],
Z = [p(np(n(stimulation), mod([pp(of, [patient], np(n(release), mod([cn([agent], n(liver)),
    cn([result], n(glucose)), pp(in, [location], np(n(blood), mod([]), ext([]))), pp(by, [manner],
    np(n(glycogenolysis), mod([]), ext([]))), pp(by, [manner], np(n(gluconeogenesis), mod([]),
    ext([]))))]), ext([]))))]), ext([])), vp(verb(active(affect), mod([pp(in, [manner], np(n(manner),
    mod([adj(opposite), pp(as, [comparison], np(n(effect), mod([rc(verb(passive(produced),
    mod([])), np(n(concentration), mod([adj(high), cn([bearer], n(insulin))]), ext([]))))]),
    ext([]))))]), ext([]))))])), np(n(concentration), mod([cn([bearer], n(blood)), cn([bearer],
    n(glucose))]), ext([]))))] .

?− sentence(250,X), p(Z,X,[]).
X = [secretion, of, insulin, into, blood, in_response_to, blood, glucose, concentration, isa,
    primary, mechanism, that, keep, within, narrow, limit, glucose, concentration, in,
    extracellular, fluid],
Z = [p(np(n(secretion), mod([pp(of, [result], np(n(insulin), mod([]), ext([]))), pp(into,
    [direction], np(n(blood), mod([]), ext([]))), pp(in_response_to, [cause], np(n(concentration),
    mod([cn([bearer], n(blood)), cn([bearer], n(glucose))]), ext([]))))]), ext([])),
    vp(verb(active(isa), mod([])), np(n(mechanism), mod([adj(primary), rc(verb(active(keep),
    mod([pp(within, [location], np(n(limit), mod([adj(narrow)]), ext([]))))])), np(n(concentration),
    mod([cn([bearer], n(glucose)), pp(in, [location], np(n(fluid), mod([adj(extracellular)]),
    ext([]))))]), ext([]))))]), ext([]))))] .
```

D.5 Graph decomposition of the sentences

The following shows the first 30 facts obtained using the graph decomposition algorithm.

```
isa([secretion,of,insulin,into,blood,in__response__to,blood,glucose,concentration],
[primary,mechanism,that,keep,within,narrow,limit,glucose,concentration,in,extracellular,fluid]).
fact(prep,[secretion,of,insulin],of,[insulin],result).
isa([secretion,of,insulin], [secretion]).
fact(prep,[secretion,into,blood],into,[blood],direction).
isa([secretion,into,blood],[secretion]).
fact(prep,[secretion,in__response__to,blood,glucose,concentration],
     in__response__to,[blood,glucose,concentration],cause).
isa([secretion,in__response__to,blood,glucose,concentration],
     [secretion]).
isa([blood,concentration],[concentration]).
fact(cn,[blood,concentration],none,[blood],[bearer]).
isa([glucose,concentration],[concentration]).
fact(cn,[glucose,concentration],none,[glucose],[bearer]).
isa([blood,glucose,concentration],[blood,concentration]).
isa([blood,glucose,concentration],[glucose,concentration]).
isa([secretion,of,insulin,into,blood,in__response__to,blood,glucose,concentration],
     [secretion,into,blood,in__response__to,blood,glucose,concentration]).
isa([secretion,of,insulin,into,blood,in__response__to,blood,glucose,concentration],
     [secretion,of,insulin,in__response__to,blood,glucose,concentration]).
isa([secretion,of,insulin,into,blood,in__response__to,blood,glucose,concentration],
     [secretion,of,insulin,into,blood]).
isa([secretion,into,blood,in__response__to,blood,glucose,concentration],
     [secretion,in__response__to,blood,glucose,concentration]).
isa([secretion,into,blood,in__response__to,blood,glucose,concentration],
     [secretion,into,blood]).
isa([secretion,of,insulin,in__response__to,blood,glucose,concentration],
     [secretion,in__response__to,blood,glucose,concentration]).
isa([secretion,of,insulin,in__response__to,blood,glucose,concentration],
     [secretion,of,insulin]).
isa([secretion,of,insulin,into,blood],[secretion,into,blood]).
isa([secretion,of,insulin,into,blood],[secretion,of,insulin]).
isa([primary,mechanism],[mechanism]).
isa([mechanism,that,keep,within,narrow,limit,glucose,concentration,in,extracellular,fluid],
     [mechanism]).
fact(rc,[mechanism],[keep],[glucose,concentration,in,extracellular,fluid],definition).
attach(fact(rc,[mechanism],[keep],[glucose,concentration,in,extracellular,fluid],definition),
     [keeping,of,glucose,concentration,in,extracellular,fluid,by,mechanism,within,narrow,limit]).
fact(prep,[concentration,in,extracellular,fluid],
     in,[extracellular,fluid],location).
isa([concentration,in,extracellular,fluid],[concentration]).
isa([extracellular,fluid],[fluid]).
isa([extracellular,fluid],[extracellular,entity]).
( ...)
```

D.6 Some examples of graph querying

D.6.1 Querying around one concept

```
?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[glucose,production,by,liver],X).
X = [every, [glucose, production, by, liver], is, a, [glucose, production]]
X = [every, [glucose, production, by, liver], is, a, [production, by, liver]]
X = [every, [glucose, production, by, liver], is, a, [production]]
X = [every, [glucose, production, by, liver], made, by, [liver]]
X = [every, [glucose, production, by, liver], is, related, by, [result], to, [glucose]]
X = [every, [glucose, production, by, liver], is, a, [glucose, production], which, is, a, [production]]
X = [every, [glucose, production, by, liver], is, a, [glucose, production], which, is, related, by,
    [result], to, [glucose]]
...

?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[glucose,production,by,liver],X).
% Error discussed in chapter 8 due to missing condition
X = [every, [betacell], [stop, at, low, glucose, concentration], by, observation,
    [insulin, secretion, into, general, circulation]]
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood],
    by, observation, [insulin]]
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood],
    by, observation, [hormone, that, is, produced, by, betacell, of, pancreatic, islet]]
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood],
    by, observation, [peptide, hormone]]
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood],
    by, observation, [peptide, hormone, that, is, produced, by, betacell, of, pancreatic, islet]]
...
```

D.6.2 Querying relation between two concepts

```
?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[betacell],[insulin],X).
X = [every, [betacell], [secrete, at, high, glucose, concentration, into, blood], by, observation,
    [insulin]]

?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[betacell],[glucose],X).
X = [some, [betacell], [secrete, at, high, glucose, concentration, into, blood], by, observation,
    [insulin], which, [regulate, by, promotion, of, absorption, of, glucose, from, blood, into, fat,
    into, liver, into, skeletal, muscle], by, observation, [metabolism, of, fat], which, involves,
    [fat], which, is, the, host, of, [conversion, of, absorbed, glucose, in, fat], which, involves,
    [glucose]]

?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[high,glucose,concentration],[synthesis],X).
X = [some, [high, glucose, concentration], is, a, [high, concentration], whereof, some, are, a,
    [high, insulin, concentration], which, is, related, by, [bearer], to, [insulin], whereof, some,
    are, a, [circulating, insulin], which, [affect], by, observation, [synthesis]]

?− read_from_text("insulin_kb.txt",KB),!, search(KB,8,[metabolism,of,protein],[blood],X).
X = [some, [metabolism, of, protein], [regulate, by, promotion, of, absorption, of, glucose, from,
    blood, into, fat, into, liver, into, skeletal, muscle], by, observation, [insulin], which, is,
    created, by, [secretion, of, insulin, at, high, glucose, concentration, into, blood], which, is,
    made, in, the, direction, of, [blood]]
```

Bibliography

[1] Troels Andreasen, Henrik Bulskov, Jørgen Fischer Nielsson, and Per Anker Jensen. “On the Relationship between a Computational Natural Logic and Natural Language”. In: *Proceedings of the 8th International Conference on Agents and Artificial Intelligence*. SciTePress. 2016 (cited on pages 11, 12, 14, 20, 27, 38).

[2] Troels Andreasen, Henrik Bulskov, Jørgen Fischer Nilsson, and Per Anker Jensen. “A System for Conceptual Pathway Finding and Deductive Querying.” In: *FQAS*. 2015, pages 461–472 (cited on pages 12–14, 70, 71, 73–76).

[3] Troels Andreasen, Henrik Bulskov, Jørgen Fischer Nilsson, and Per Anker Jensen. “Computing Pathways in Bio-Models Derived from Bio-Science Text Sources.” In: *IWBBIO*. 2014, pages 217–226 (cited on pages 4, 8, 12, 73, 75).

[4] Troels Andreasen, Henrik Bulskov Styltsvig, Per Anker Jensen, and Jørgen Fischer Nilsson. “A Natural Logic for Natural-Language Knowledge Bases”. In: *Partiality and Underspecification in Information, Languages, and Knowledge*. Cambridge Scholars Publishing, 2016 (cited on pages 12, 14, 27, 38, 39).

[5] Per Anker. “A note on salient syntactic structures involving event verbs in biomedical texts”. unpublished note. 2017 (cited on page 30).

[6] Robert Arp and Barry Smith. “Function, role and disposition in basic formal ontology”. In: (2008) (cited on page 52).

[7] Gilad Ben-Avi and Nissim Francez. “Categorial grammar with ontology-refined types”. In: *Proc. Categorial Grammars–An efficient tool for Natural Language Processing* (2004) (cited on pages 50, 58, 61).

[8] Patrick Blackburn and Johan Bos. “Computational semantics for natural language”. In: *Course notes for NASSLLI* (2003) (cited on page 10).

[9] Patrick Blackburn, Johannes Bos, and Kristina Striegnitz. *Learn prolog now!* Volume 7. 7. College Publications London, 2006 (cited on page 95).

[10] Peer F Bundgaard, Svend Østergaard, and Frederik Stjernfelt. “Meaning construction in the production and interpretation of compounds is schema-driven conceptual schemata and cognitive operations in compound constructions”. In: *Acta Linguistica Hafniensia* 39.1 (2007), pages 155–177 (cited on page 22).

[11] Elizabeth Coppock. “The Predicativity Principle”. 10th Annual Semantics Fest, Stanford University. 2009. URL: `http://www.eecoppock.info/CoppockSemFest09.pdf` (cited on page 26).

[12] Michael A Covington. “Natural language plurals in logic programming queries”. In: (1996) (cited on page 37).

[13] Ernest Davis (New York University). *Notes on Ambiguity*. URL: `http://cs.nyu.edu/faculty/davise/ai/ambiguity.html` (cited on page 48).

[14] Ernest Davis (New York University), Leora Morgenstern, Charles Ortiz. *The Winograd Schema Challenge*. URL: `http://www.cs.nyu.edu/faculty/davise/papers/WinogradSchemas/WS.html` (cited on page 48).

[15] Enrico Franconi. “A treatment of plurals and plural quantifications based on a theory of collections”. In: *Minds and machines* 3.4 (1993), pages 453–474 (cited on pages 36, 37).

[16] Lyn Frazier. “On comprehending sentences: Syntactic parsing strategies”. In: (1979) (cited on page 49).

[17] Lauren A Fromont, Salvador Soto-Faraco, and Emmanuel Biau. “Searching high and low: prosodic breaks disambiguate relative clauses”. In: *Frontiers in Psychology* 8 (2017) (cited on page 49).

[18] Richard A Frost, Rahmatullah Hafiz, and Paul Callaghan. “Parser combinators for ambiguous left-recursive grammars”. In: *International Symposium on Practical Aspects of Declarative Languages*. Springer. 2008, pages 167–181 (cited on page 18).

[19] Ulrich Furbach, Ingo Glöckner, Hermann Helbig, and Björn Pelzer. “Logic-based question answering”. In: *KI-Künstliche Intelligenz* 24.1 (2010), pages 51–55 (cited on page 1).

[20] Benjamin N Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. “Description logic programs: combining logic programs with description logic”. In: *Proceedings of the 12th international conference on World Wide Web*. ACM. 2003, pages 48–57 (cited on page 5).

[21] Nicola Guarino et al. “Formal ontology and information systems”. In: *Proceedings of FOIS*. Volume 98. 1998. 1998, pages 81–97 (cited on pages 3, 52).

[22] John Hale. “Sentence Processing: ambiguity”. 2003. URL: `https://msu.edu/course/lin/401/fs03-s2/sp-lecture1.pdf`. lecture from the course, introduction to linguistic theory (cited on page 47).

[23] Heinrich Herre. “General Formal Ontology (GFO): A foundational ontology for conceptual modelling”. In: *Theory and applications of ontology: computer applications*. Springer, 2010, pages 297–345 (cited on page 52).

[24] Jeff Speaks (University of Notre Dame). *Modifiers*. URL: `http://www3.nd.edu/~jspeaks/courses/2012-13/43916/handouts/13-modifiers.pdf` (cited on page 25).

[25] Per Anker Jensen and Jørgen Fischer Nilsson. “Ontology-based semantics for prepositions”. In: *Syntax and Semantics of Prepositions*. Springer, 2006, pages 229–244 (cited on pages 51, 53, 87).

[26] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. “A description logic primer”. In: *arXiv preprint arXiv:1201.4089* (2012) (cited on pages 5, 7).

[27] Bill MacCartney and Christopher D Manning. "Natural logic for textual inference". In: *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing.* Association for Computational Linguistics. 2007, pages 193–200 (cited on page 13).

[28] Marcin Morzycki. *Modification.* Cambridge University Press, 2015 (cited on page 32).

[29] Neha Nayak, Mark Kowarsky, Gabor Angeli, and Christopher D Manning. *A dictionary of nonsubsective adjectives.* Technical report. Technical Report CSTR 2014-04, Department of Computer Science, Stanford University, October, 2014 (cited on page 25).

[30] Jørgen Fischer Nilsson. "A note on the puzzling Relationship between Affirmative Natural Logic and Description Logic". draft article. 2017 (cited on page 44).

[31] Jørgen Fischer Nilsson. "In pursuit of natural logics for ontology-structured knowledge bases". In: *7th International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 2015).* 2015, pages 42–46 (cited on pages 4, 6, 7, 12).

[32] Natalya F Noy, Deborah L McGuinness, et al. *Ontology development 101: A guide to creating your first ontology.* 2001 (cited on page 3).

[33] Nick G Riches, Tom Loucas, Gillian Baird, Tony Charman, and Emily Simonoff. "Elephants in pyjamas: Testing the weak central coherence account of autism spectrum disorders using a syntactic disambiguation task". In: *Journal of autism and developmental disorders* 46.1 (2016), pages 155–163 (cited on page 49).

[34] Gillian Russell. *The Routledge companion to philosophy of language.* Routledge, 2012 (cited on page 25).

[35] Stuart Russell, Peter Norvig, and Artificial Intelligence. "A modern approach". In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25 (1995), page 27 (cited on page 10).

[36] Barry Smith. "International Conference on Biomedical Ontology in Lisbon". In: 2015 (cited on page 53).

[37] Barry Smith, Anand Kumar, and Thomas Bittner. "Basic formal ontology for bioinformatics". In: *Journal of Information Systems* (2005), pages 1–16 (cited on page 52).

[38] Wikipedia. *Axiom of extensionality — Wikipedia, The Free Encyclopedia.* [Online; accessed 22-May-2017]. 2017. URL: `%5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Axiom_of_extensionality&oldid=778703565%7D` (cited on page 37).

[39] Wikipedia. *Insulin — Wikipedia, The Free Encyclopedia.* [Online; accessed 16-November-2016]. 2016. URL: `%5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Insulin&oldid=779594959%7D` (cited on pages 21, 35, 45, 47, 53, 55, 59, 89, 91).

[40] Wikipedia. *Paraphrase — Wikipedia, The Free Encyclopedia.* [Online; accessed 20-June-2017]. 2017. URL: `%5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Paraphrase&oldid=785028395%7D` (cited on page 26).

[41] Wikipedia. *Plural quantification — Wikipedia, The Free Encyclopedia.* [Online; accessed 22-February-2017]. 2016. URL: `%5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Plural_quantification&oldid=747616885%7D` (cited on page 37).

[42] Wikipedia. *Thematic relation — Wikipedia, The Free Encyclopedia.* [Online; accessed 17-May-2017]. 2017. URL: `%5Curl%7Bhttps://en.wikipedia.org/w/index.php?title=Thematic_relation&oldid=770782480%7D` (cited on page 55).