Semantic Querying with Ontologies

Saba Kashan Fallah

Kongens Lyngby 2007
IMM-M.Sc.-2007-89

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

In this project the usage of syntactic-semantic lexical resources like FrameNet combined with formal ontologies for the purpose of semantic interpretation of domain specific text, is researched. Accurate ontology driven semantic interpretation of natural language is only possible in terms of mappings based on reliable linguistic semantics, that maps linguistic forms to ontological types. The novelty of the method proposed in this report is to extend frame semantics with formal domain specific ontological types, in order to achieve deep semantic interpretation of text. The aim is deep semantic interpretation of scientific biomedical text, which in contrast to shallow semantic interpretation is not based on linguistic semantics only. Semantics of text is represented in terms of ontological concepts/classes, and frame-based definitions/descriptors. The twofold semantic representation provides not only mappings from text to nodes in ontological hierarchies and reverse, but also a mapping from query descriptors to ontological concepts and vice versa. Semantic annotations provided in the form of descriptors are used in order to semantically search text, where descriptors of user queries are matched against text-descriptors. Real ontological information retrieval is implemented in terms of descriptor subsumption based on the ontological hierarchies. Search based on the ontological hierarchy or input queries or a combination of both is made possible. The ontological structures are incorporated in the system, not only as a collection of terms but true to their nature as semantic networks. Methodologies and techniques for implementation of the logical core of such a multi-dimensional system, is suggested along with a proof-of-concept demo.

Key words: semantic interpretation, formal ontologies, frame semantics, ontology driven information retrieval, frame-based grammar, ontological grammar,

parsing techniques.

Resumé

I denne projekt, brugen af syntaktisk-semantisk leksikalske ressourcer såsom FrameNet kombineret med formelle ontologier, med henblik på semantisk fortolkning af domæne specifikke tekster, er undersøgt. Præcis ontologi baserede semantiske fortolkninger af naturligt sprog er kun muligt, baseret på afbildninger ("mappings") fra pålidelige lingvistisk baseret semantik, der overfører lingvistiske mønstre til ontologiske typer. Det nyskabende ved metoden fremlagt i denne rapport, er at udvide frame-semantik med formelle domæne specifik ontologi baseret semantiske typer, for at kunne opnå en dybtgående semantik fortolkning af tekst. Formålet er en dybtgående semantisk fortolkning af videnskabelige biomedicinske tekster, der i modsætning til en overfladisk semantisk fortolkning, er ikke kun baseret på lingvistisk semantik. Formel semantik repræsenteret i form af ontologiske termer og frame baserede klasse definitioner/beskriver er brugt til at repræsentere teksters semantik. Den dobbelt semantiske repræsentation bidrager ikke alene med afbildninger fra tekst stykker til knuder i ontologiske hierarkier og omvendt, men derimod også med afbildninger af forespørgsel beskriver til ontologiske koncepter og omvendt. Semantiske annotationer i form af beskriver er brugt for at kunne søge tekst semantisk, hvor beskriver af brugerens forespørgsler er forenet med beskriver fra tekster. Søgninger baseret på det ontologiske hierarki eller via forespørgsler, eller en kombination af begge er blevet gjort muligt. De ontologiske strukturer er inkorporeret i systemet, hvor ontologier er ikke alene brugt som en samling af termer men tro mod deres nature som semantiske netværker. Metodologier og teknikker for implementering af den logiske kerne af sådan et multidimensional system er blevet forsålet, samt med at en proof-of-concept demo er implementeret.

Nøgleord: semantisk fortolkning, formelle ontologier, frame-semantik, ontologi baseret informations søgning, frame baseret grammatik, ontologisk grammatik,

parsing teknikker.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in fulfillment of the requirements for acquiring the M.Sc. degree in engineering. It was prepared during spring and summer 2007 and has been supervised by Professor Jørgen Fischer Nilsson.

The thesis deals with the topic of domain specific ontology-based semantic interpretation of domain text, based on the notion of semantic frames. The main focus is evaluation and demonstration of a semantic parser based on a ontologically constrained frame-based grammar.

I would like to thank everyone who has has help me with this thesis, especially Jørgen Fischer Nilsson for his guidance and supervision. And I will further tank my girlfriend and family for their support.

Lyngby, September 2007

Saba Kashan Fallah

Contents

Chapter 1

Introduction

This project has been a study of ontology-based semantic interpretation of domain specific text, and how it can be used in ontology driven information retrieval. Semantic interpretations are based on the idea of frame semantics, which is a method of describing meaning of linguistic expressions via structures called frames. Frames, that model the semantic structures of events, are used in order to analyze and represent meaning of text.

In order to facilitate ontology-based semantic interpretation, these frame structures must be constrained with ontological types. Since frames in domain ontology-driven semantic interpretation, are supposed to describe events in the considered domain of discourse, the frame-elements participating in these events must satisfy conditions set with regard to their ontological types. This way of constraining frames and their elements, is necessary in order to be able to determine and verify the events modeled by frames, and expressed in linguistic structures (text). Further frames extended with ontological type-constrains are essential for being able to map linguistic expressions to corresponding, formally defined compound concepts in domain ontologies.

In the following the concept of frame semantics is introduced in addition to an introduction of formal ontologies and the way they are used in computer system for e.g. the purpose of semantic interpretation. We will further introduce the notion of ontology-driven information retrieval.

A Brief introduction of frame semantics with regard to semantic interpretation is introduced in Section 1.1. Section 1.2 introduces formal ontologies and their role in computer systems. In Section 1.3 the concept of ontology-driven information retrieval is elaborated on. An overview on how this document is structured, and what to except from the content is given in Section 1.4.

1.1 Semantic-interpretation and Semantic Frames

Frame Semantics is a way of analyzing and describing meaning of natural language. Semantics in natural language is the analysis of linguistic meaning, based on the conventions of the language. Linguistic semantics is an account of the syntactic-structures of the language and the corresponding semantic-decomposition. I.e. it concerns with how the semantics is related to syntax.

Every linguistic expression is a reference to a *situation* or an *event*. Units in semantic analysis are these events in relation with their constituents (entities that participate in the events). The concept of frame semantics is concerned with analysis and presentation of semantics of linguistic expression with respect to certain underlying events. Structures called frames are used to specify events and situations.

The concept of frame semantics is a linguistics approach to semantics, that is based on empirical evidence in large collection of text. The idea is to extend this linguistics founded semantic structures, with controlled vocabularies of formal domain ontology, in order to be able to map form (text) to semantics (ontological concepts). The motivation is in particular, to be able to recognize and analyze natural language realizations (descriptions) of real life events and map them to the corresponding ontological terms.

1.2 Formal Ontologies

Ontologies are included in computer systems with the purpose of modeling and representation of domain knowledge in a structured computable format. The formally structured knowledge in this way, can be used for reasoning, and for generation, and querying of tacit information. These tacit information, that can be induced are e.g. hierarchial relations between objects/entities or other kinds of relational properties. This is in contrast to systems that are based on explicit representations of knowledge.

The concept of hierarchically classifying knowledge about a domain, is inspired from philosophy, where philosophers for years have sought to categorize the reality. The aim of using ontologies in information systems is not to give an account of reality, but to represent a certain view on a domain (a particular portion of reality), in a formal computer-readable format. Here in this project a formal representation of the domain of biological processes in the form of an ontology (Gene Ontology), lays the basis for a semantic parser.

Gene Ontology represents the shared conceptualization of bio-medical scientist, consisting of a collection of concepts and axiomatized relations among these. The vocabulary established by GO, represent how bio-medical science view and express issues concerning biological processes. These terms represents objects and phenomena that scientists believe exist. The ontology (as well as our system) is committed to this vocabulary/terminology, used by scientist to describe events in the field of bio-processes. The role of ontology in this sense is to formalize this knowledge.

For the formalization a formal language like first-order logic is preferred, because of the clear semantics and its deductive properties. Ontologies essentially are hierarchical categorization of a set of entities that are related to each other in a series of axiomatized relations. Axiomatized relations, means that the relations are defined in terms of a rigorous formal language with clear semantics.

The backbone (skeleton) of each formal ontology is a taxonomy. Taxonomy is the hierarchy established by categorizing entities with regard to their shared properties. In order for any ontology based reasoning system to be consistent, the considered ontology must be consistent itself. In particular the taxonomy of the ontology must not contain any contradiction. Methodologies for eliminating contradictions in ontologies are developed, one of which is OntoClean (see [**?**] for more details). Consistent taxonomies are type hierarchies that are not only used in semantic evaluation but also assist to meaningful semantic based search and querying.

1.3 Ontology-driven Information Retrieval

Ontology-driven information retrieval is the idea of searching information in e.g. text based on the semantics of the content. This is in contrast to conventional key-word search which traditionally is based on occurrence of key-words in the content and their distance.

The idea with ontology-driven or ontology assisted search, is to use the hierar-

chically classified knowledge represented by ontologies in order to semantically place information present in content. The content will be marked up (annotated) in terms of contextual knowledge provided by the ontology. This method of ontology-based semantic annotation provides a whole range of tacit information, that are generalizations and specializations of information present in text; or are contextually related to the information explicitly present in the content.

Concretely, the method consists of interpreting pieces of information in terms of their place in classification hierarchies of ontologies, i.e. identifying the ontological concepts corresponding to pieces of information. This is done by annotating the semantics of the content via semantic structures that define the underlying concept. This places the concepts of the content in the context established by the ontologies.

Queries can be interpreted in the same way, and based on their place in the hierarchy (ontology) matched with related concepts present in text. This basically means that by placing information in ontological hierarchies we can answers queries seeking some concept with more specific of general concept described i text.

1.4 Structuring of the Material

Here follows an overview of this document. The subjects covered in each chapter are briefly described.

Chapter 1 and 2 concern with the idea of frame semantics, and cover the linguistics theories and phenomena that help us to understand, the concept and related issues. Chapter 1 will start with an introduction of Context Free Grammars (CFG) and some properties of natural languages that are not easily modeled by CFG are accounted for. This is used as a starting point of a discursion on the motivation for semantic frames. Chapter 2 will introduce the concept of frame semantics in details. A long with introducing the pertaining terminology, concrete examples from the bio-medical domain has been considered in attempt to, show what the real nature of frames are, and how they can be used in semantic interpretation.

Chapter 4 is a full-scale study (tutorial) on different techniques of parsing. By considering different techniques and implementing some of these in Prolog; and covering a discussion on each technique's weaknesses and strength; the attempt has been to justify the choice of technology made for the implementation of a small demo.

Chapter 5 is a study on design of a semantic parser, based on frames extended with formal ontologies. The systems and its properties in terms of requirements and facilities are discussed, accompanied with an overview on the process of identifying ontology based frames.

Chapter 2

Linguistic Theory and Formal Grammars

This chapter introduces the notion of *generative grammars* that is accompanied with linguistic theories and notions. Knowledge of these notions will help understanding, the concepts of frames and frame semantics, which is key to the method suggested in this project. Further it is crucial to understand these linguistic notions in relation to formal grammars of natural languages, in order to understand why lexical-resources like FrameNet are required in order to achieve a sufficiently deep semantic interpretation.

We start by introducing formal grammar in Section 2.1.Section 2.2 covers some linguistic notions that clarify interesting properties of natural language.

2.1 Formal Grammar

In the following the mathematical definition of a context-free grammar is given, accompanied by corresponding terminology used in linguistics. An example of a grammar is included which is a simple grammar of English. This simple English grammar will set the stage for introducing some interesting linguistic properties. Further we will discuss the notion of generative-grammar, that denotes the facts

that formal grammars as formalizations, are used more extensively then just analysis of syntax.

2.1.1 Context-Free Grammars

A Context-Free Grammar (CFG) recursively defines a language. Recursively defines a language means that syntactical-categories in the grammar are defined in terms of other syntactical-categories that are in return defined by others themselves. The language of a grammar is all acceptable strings of words (symbols) that can be predicted by the grammar.

A context-free grammar is a formal system consisting of the following four components:[11]

Terminals: a finite set of words (symbols) that any sentence of the language is formed of. Correspondingly a sentence (string) of the language is a sequence of these terminals. In the linguistic context *terminals* are referred to as the lexicon where words are grouped into lexical categories (part of speech classes) such as noun, verbs and adjectives.[10]

Nonterminals: a finite set of variables or syntactic-categories that each represent a phrase/structure-type in the language. In the linguistic context nonterminals are referred to as phrasal categories or non-lexical categories.[10]

Start Symbol: is the variable (nonterminal) that defines all well-formed sentences of the language. It is the entry-point of the language, that via its sub-phrases(other nonterminals in the grammar) stipulates the valid sentences of the language.

Production Rules: these are structures that recursively (via other nonterminals) stipulate the valid patterns of a syntactic-category (phrasal-category). A production-rule is a mathematical structure of the following form $A \rightarrow \varphi$, where A (a nonterminal) is the *head* or (as is preferred here) the left-handside of the production rule and φ is the *body* or right-handside. The head represents a valid structure that is defined by the body. The body is a ordered set (a sequence) of terminals and nonterminals. Same nonterminal can be the head of a series of production rules that each define an alternative for the syntactic pattern of the head.

The idea of using grammar to formalize a language is based on the notion of *Constituency*. Constituency is concerned with how words group together to build phrases that can in return be constituents of greater phrases. In the English

$\langle sent \rangle$	$\rightarrow \langle np \rangle \langle vp \rangle$
$\langle np \rangle$	$\rightarrow (\langle det \rangle)\ \langle nom \rangle$
$\langle np \rangle$	$\rightarrow \langle adj \rangle\ \langle np \rangle$
$\langle nom \rangle$	$\rightarrow \langle noun \rangle$
$\langle nom \rangle$	$\rightarrow \langle nom \rangle \langle pp \rangle$
$\langle vp \rangle$	$\rightarrow \langle verb \rangle \langle pp \rangle$
$\langle pp \rangle$	$\rightarrow \langle prep \rangle \langle np \rangle$
$\langle verb \rangle$	$\rightarrow$forces
$\langle noun \rangle$	$\rightarrow$regulation|storage|glucose|cells
$\langle prep \rangle$	$\rightarrow$of|in
$\langle adj \rangle$	$\rightarrow$liver

Table 2.1: Simple English Grammar

grammar of Table 2.1, every acceptable English sentence has two constituents a *noun-phrase* followed by *verb-phrase*, and each of the constituents are defined in terms of their own constituents (sub-phrases). Further the lexical parts of the grammar (lexical-categories) consists of *nouns*, *verbs* and *preps* (prepositions). The grammar is not complete due to the limitations of the attached lexicon and due to deficiencies in terms of rules handling e.g. subject-verb agreement restrictions and other restrictions in general. Additionally it is a fact that the grammar over-generates, i.e. it generates inacceptable English sentence as well as acceptable sentences.[10]

With the notion of CFG in place we can turn our focus on the more general term of generative-grammars. The term *Generative-grammar* refers to recursive formalizations similar to CFG, but is more general then the notion of CFG. The term generative-grammar is preferred over CFG, as generative-ness and rule-based characteristics of grammars, as formalizations are used extensively in e.g. semantic-parsers or other NLP(Natural Language Processing) systems. Generative-grammars can be formal systems of rules that e.g. stipulate the correct syntactic structures and the corresponding semantics of phrases in a language; or stipulate relations among terms in a controlled vocabulary (terminology established by an ontology). Generally the notion generative-grammar emphasizes the fact that a formal grammar of a language not only determines the well-formedness of a language but also can be used to generate deduce consistent data.[10]

2.2 Linguistic Properties and Notions

In this section some linguistic notions are briefly introduced. In particular the notion of headed phrases and the related notions of valence are explained. These notions express linguistic properties that are semantically significant. I.e. no semantic analysis without attention to these properties will be sound. In our attempt to introduce these, we will start by looking at some simple restrictions on patterns of syntactic structures. Further attribute-value pair matrices called *Feature-Structures* (FS) are used in order to represent the material.

2.3 Co-occurrence Restrictions

Co-occurrence restrictions sanction what words can go together. Co-occurrence comprise consist among others of transitivity and agreement. Transitivity is important for us, since it is semantically dependent. An examples illustrating this point is included below. The first sentence (1) is not a valid sentence since, the verb *force* is transitive, i.e. it must be followed by a noun-phrase. The transitiveness of the verb is in accordance to its meaning here, which is causing something. Second sentence on the other hand does not violate the transitivity restriction of the verb force, and is a meaningful acceptable English sentence.

1. *Insulin forces.
2. Insulin forces storage of glucose.

In the following we will use FS notation in order to introduce linguistic phenomena like *transitivity* and *agreement* etc. Feature-structures are used in order to represent syntactic and semantic information of lexical-entries (words). Similarities (parallelism) in structures of different phrase-types are generalized in FS-extended grammars like in Table 2.2.

Table 2.2 contains a set of FS-extended rules that handel the issues of transitivity and agreement. Considering the three last rules b,c and d (in Table 2.2), transitivity is stipulated by the patterns on righthand-sides, and in terms of the values of the feature *VAL* (abbreviation of valence) of the left-most constituent. Agreement which is concerned primarily with subject-object agreement is partially covered in the rules by identity of the *NUM* feature values.

FS-extended grammatical rules of the kind included in Table 2.2 are based on the *unification* operation FS.(See [10] for more details on unification.)

a.

$$\text{S}\rightarrow\begin{bmatrix}\text{phrase}\\ \text{POS} & \text{noun}\\ \text{NUM} & \boxed{1}\end{bmatrix}\begin{bmatrix}\text{phrase}\\ \text{POS} & \text{verb}\\ \text{NUM} & \boxed{1}\end{bmatrix}$$

b.

$$\begin{bmatrix}\text{phrase}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\end{bmatrix}\rightarrow\begin{bmatrix}\text{word}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\\ \text{VAL} & \text{itr}\end{bmatrix}$$

c.

$$\begin{bmatrix}\text{phrase}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\end{bmatrix}\rightarrow\begin{bmatrix}\text{word}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\\ \text{VAL} & \text{tr}\end{bmatrix}\text{NP}$$

d.

$$\begin{bmatrix}\text{phrase}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\end{bmatrix}\rightarrow\begin{bmatrix}\text{word}\\ \text{POS} & \boxed{1}\\ \text{NUM} & \boxed{2}\\ \text{VAL} & \text{dtr}\end{bmatrix}\text{NP NP}$$

Table 2.2: Rules covering Transitivity and Agreement [10]

b.
$$\left[\text{phrase}\right] \rightarrow \text{H}\begin{bmatrix}\text{word} & \\ \text{VAL} & \text{itr}\end{bmatrix}$$
c.
$$\left[\text{phrase}\right] \rightarrow \text{H}\begin{bmatrix}\text{word} & \\ \text{VAL} & \text{tr}\end{bmatrix}\text{NP}$$
d.
$$\left[\text{phrase}\right] \rightarrow \text{H}\begin{bmatrix}\text{word} & \\ \text{VAL} & \text{dtr}\end{bmatrix}\text{NP NP}$$

Table 2.3: Head Rules
[10]

2.3.1 Headed Phrases and Valence

In order to understand co-occurrence restriction properties in general and valence in particular, we have to consider the notion of *headed phrases*.

The notion of head is a way of indicating the fundamental relation a phrasal-category has to one of its lexical-constituent. The structures of phrasal-categories (e.g. NP or VP) dependent on the syntactic and semantic properties of one of their lexical constituents, namely the *head*. Syntactic patterns and the corresponding semantics of headed phrases are determined by their lexical heads. This is expressed by the head-feature principle that follows below.

Head-Feature Principle:
Any headed-phrase inherits the features of its lexical head.[10]

In English, phrasal-categories are mostly governed or headed, by their left-most lexical constituents.[10] The grammatical rules of Table 2.2 show how the heads - the left-most constituents - dictate the features and structure of the phrases, in terms of transitivity and agreement. With this in mind and the head-feature principle at disposal, the three last rules (a),(b) and (c) from Table 2.2 can be generalized to the rules in Table 2.3.

We will now take a look at the more relevant notions of valence and complement. *Complement* is the term used for phrase-types that may occur after a head, in a headed-phrase structure. *Valence* is the combinatory possibilities or sanctioned patterns of the complements.[10] A phrase usually consists of the head and the heads complements. This is what formally is expressed below as the head-complement rule.

Head-Complement Rule:

$$\begin{bmatrix}\text{phrase}\\ \text{COMPS} & \langle\,\rangle\end{bmatrix} \rightarrow \text{H} \begin{bmatrix}\text{word}\\ \text{COMPS} & \langle \boxed{1} \ldots \boxed{n} \rangle\end{bmatrix} \boxed{1} \ldots \boxed{n}$$

Since valences of verbs mostly dependent on the verbs' semantics, and vice versa; the notions of valence and head-ness are very important in semantic analysis of linguistic expressions in general, and for frame-based semantic-parsing in particular. In frame-based semantic processing the underlying grammar is based on case-based rules that express valence patterns of words, mostly verbs. Lexical resources like FrameNet consist mainly of valence patterns for words in English. But valence patterns included in e.g. FrameNet comprise more than just the complements of the words they also include e.g. specifier along with the head and complements.

There are co-occurrence restriction concerning what may co-occur with a word positioned before the word. Specifiers are words that may come before a e.g. verb in a sentence. Specifiers restrictions are concerned with non-complement co-occurrence. The most common examples of specifiers are subjects of verbs and determiners of nouns. The rule below expresses specifier constrain of phrases.

Head-Specifier Rule:

$$\begin{bmatrix}\text{phrase}\\ \text{SPR} & \langle\,\rangle\end{bmatrix} \rightarrow \boxed{1} \quad \text{H} \begin{bmatrix}\text{word}\\ \text{SPR} & \langle \boxed{1} \rangle\end{bmatrix}$$

Complete valence-patterns of verbs (in e.g. FrameNet) convey information on the specifier constraints of the verb along with lists of its complements. An example of such valence-pattern for the verb is given below.

$$\left\langle \text{force}, \begin{bmatrix}\text{HEAD} & \text{verb}\\ \text{SPR} & \left\langle \begin{bmatrix}\text{HEAD} & [\text{noun}]\end{bmatrix} \right\rangle\\ \text{COMPS} & \left\langle \begin{bmatrix}\text{phrase}\\ \text{HEAD} & \text{noun}\end{bmatrix} \right\rangle\end{bmatrix} \right\rangle$$

Figure 2.1 illustrate head-complement and head-specifier rules in action, with

regard to a verb-phrase headed by the verb force.

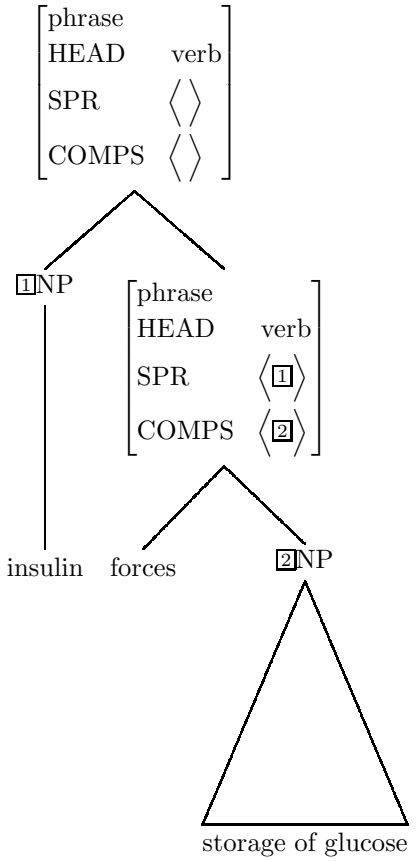


Figure 2.1: Valence Dictated by the Lexical-entry *force*

With these linguistic notions describing some interesting properties of natural languages, we would be better equipped for a realistic semantic analysis. What we have to realize from this is that linguistic properties are not easily modeled by CFG; and that a realistic formalization of natural languages are case-based; i.e. based on words that head or govern phrasal-structures. Further note that with among others HFP in place natural languages can still be formalized with general grammars that rely on lexicons. These lexicons present syntactic and the corresponding semantic properties of words in a consistent format like e.g. FS. Such a lexicon is e.g. FrameNet.[4]

Chapter 3

Frame Semantics

In this chapter the notion of *semantic frames* will be introduced. First linguistic semantics in general is introduced; then the concept of frame semantics as semantic structures, used with the aim of determining and representing meaning in natural language, is presented. This lays the basis for understanding the notion of frames, and why and how they are used in semantic parsing (interpretation).

Section 3.1 is a brief introduction of linguistic semantics. In section 3.2 the concept of semantic frames is introduced accompanied by the notions of semantic-roles, lexical-units, valence and valence-patterns which are illustrated by examples. In section 3.3 relations among frames and elements of these are explained. Section 3.4 is a brief description on the FrameNet project.

3.1 Linguistic Semantics

The study of semantics with respect to natural languages is concerned with the semantics of the individual words and how these combine to build the semantics of phrases. I.e. semantics of a phrase is decomposed into the semantics of its constituents.

Linguistic propositional meaning, is about structures for representing linguis-

tic meaning of words and establishing constraints (conditions) that predict semantics of phrases in terms of semantic contributions of their immediate constituents.[10] Below the constraints/conditions that predict the semantics of the proposition *"insulin forces storage of glucose"* are listed.

event(CAUSATION)
event-participators(AGENT(a),EFFECT(e))
scene(AGENT(*insulin*) causes EFFECT(*storage of glucose*))

The verb *force* establishes/predicts a *causes* relation between insulin and "storage of glucose"; insulin takes the role of an *agent* and "storage of glucose" the role of the *effect*. The *scene* describes the general situation where a certain agent causes an effect. These all together represent a *causation* event.

3.2 The Concept of Frame Semantics

Frame Semantics is a way of analyzing and describing meaning of natural language. Semantics in natural language is the analysis of linguistic meaning based on the conventions of the language. I.e. linguistic semantics is an account of the syntactic structures and the corresponding semantics of these.

Every linguistic expression is a reference to a situation or an event.[10] The focus of semantic analysis is on these events and the pertaining relations with their elements (entities that participate in the events). The concept of *frame semantics* is concerned with analysis and presentation of semantics of linguistic expressions with respect to underlying abstract events. Structures called frames are used to specify events/situations.

A *frame* is a structure consisting of a series of relations that link concepts (ontological-classes) to the frame, in order to constitute more complex concept denoted by the frame. A frame-structure is a way of representing an real event.

Prior to introducing frames and frame-annotation consider the semantic-structure illustrated in Figure 3.1. It is a network consisting of entities and relations among them. This semantic-structure depicts the event of *"insulin causing storage of glucose in liver-cells"*. Each relation (edge in the graph) denotes the role played by an participating entity (entity involved in the event). While *insulin* is the *actor* that participates (has a leading part) in the cause of the event, *insulin-stimulus* in the organism is the actual *cause* of the event; *storage-of-glucose* is the *effect* of the event while *liver-cells* are the *effected* entities.

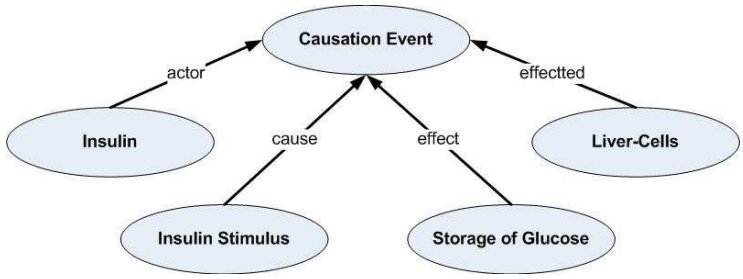


Figure 3.1: Semantic Structure of Causation Event

While a semantic structure as in Figure 3.1 is general in its outlook on the event and specifies all constituents (participating entities with respect to their roles) frame annotations can be more restricted in their outlook and only partially specify the event.

What is the nature of frame specifications, and how frame specifications are to be represented, is relevant for our understanding of frames. By taking up the discussion, we would emphasize on how frames are used to represent meaning of linguistic structures, based on the outlook a certain governing word (e.g. verbs) has on the underlying event. I.e. we want to emphasize here, that the way meaning is represented via frames depends on the frame-evoking word, i.e. the head of the phrase which represents the linguistic realization of the event.

Denoting the same Causation Event of Figure 3.1 is possible in terms of a first-order logic predicate. (3.1) is a predicate representing the causation event. Each position in the predicate (3.1) represents a role in the event, which is played by the argument. A more detailed representation of the causation-event is represented by (3.2), where each role is represented by a functor-term. These are examples of general predictions. But the key point here is that, different words with the same underlying meaning represented by a frame, have different predictions.

$$\text{causation}(\text{insulin}, \text{insulin\_stimulus}, \text{storage\_of\_glucose}, \text{liver\_cells}) \tag{3.1}$$

$$\text{causation}(\text{actor}(\text{insulin}), \text{cause}(\text{insulin\_stimulus}), \text{effect}(\text{storage\_of\_glucose}), \text{effected}(\text{liver\_cells})) \tag{3.2}$$

The idea of Frame Semantics (like first-order predicates with variables as arguments) is to denote the abstract structure of e.g. causation events. Frames (and frame-based annotation) can be represented by predicates but semantic-structures (as in 3.1) are more suitable. This is due to the fact that frames are

thought as abstract structures representing the underlying meanings of linguistic (natural language) expressions. Since events can be described in different ways and looked at from different outlooks, annotations and the underlying abstraction can have different forms and combinations.

Semantic annotation of an expression depending on the evoking word, can e.g. be a structure like (3.3). (3.4) is frame-based semantic annotation of a different expression with a different evoking word, but with the same underlying frame as in (3.3). Essentially different combinatory possibilities comes about when different words and the corresponding sentences present different outlook on the same event.

$$[_{\text{Actor}}\text{insulin}]\textbf{forces}[_{\text{Effect}}\text{storage of glucose}][_{\text{Effected}}\text{liver-cells}] \tag{3.3}$$

$$[_{\text{Cause}}\text{insulin-stimulus response}]\textbf{causes}[_{\text{Effect}}\text{storage of glucose}][_{\text{Effected}}\text{liver-cells}] \tag{3.4}$$

Frame specification is equivalent to feature-structures descriptions, used by linguists to give an account of valence and agreement of phrases and semantics as we saw in the previous chapter.

3.2.1 Semantic-Roles or Frame-Elements

A frame from our perspective is a structure for annotation and representation of linguistic appearances of events in text. A frame represents the semantical structure of an event, in terms of participators of the event by means of *frame-elements*.

Frames are compound-concepts describing abstract events. *Semantic-roles* are constitutes of these compound-concepts. Semantic-roles denote the role played by an entity participating in a particular event. A semantic role is understood in the context of the abstract event represented by the frame; similarly a frame is defined and understood by means of its elements, i.e. the corresponding semantic roles. *Frame-elements* represent the semantic-roles of an event denoted by a frame.

Table 3.1 is the definition of Causation frame accompanied by definitions of the pertaining frame-elements.

In order to illustrate the concept of semantic-roles a biological event is considered. Responsive-events are biological processes (or functions) caused by a certain stimulus, e.g. insulin-stimulus in. Table 3.2 shows the term and definition of a concept from OBO.GO-ontology, which represent a biological-process

Frame	**Frame Definition**
Causation	A *Cause* causes an *Effect*. Alternatively, an *Actor*, a participant of a (implicit) Cause, may stand in for the Cause.
Frame-element	**Frame-element Definition**
Actor	An entity which participates in a Cause.
Cause	An animate or inanimate entity, a force, or event that produces an effect.
Effect	A positive or negative evaluation of the Phenomenon.
Effected	Agents in a joint or reciprocal action.

Table 3.1: Causation Frame
[4]

Concept-term	insulin-responsive hydrogen:glucose symporter activity
Definition	Catalysis of the transfer of a solute or solutes from one side of a membrane to the other according to the reaction: glucose + H+ = glucose + H+, in response to a stimulus by insulin.

Table 3.2: Insulin-Responsive Event
[5]

caused by insulin-stimulus. In the following the event described in the Table 3.2 will be referred to as insulin-responsive event.

The insulin-responsive event is analyzed here within the general (not biological domain specific) causation-frame. Within causation-frame *Actor*(see Table 3.1) represent the role played by an entity in the *Cause* of an event. Insulin participates in the event, that causes the considered insulin-responsive event, namely the event of insulin-stimulus. In insulin-response the *Cause* of the event is the insulin stimulus-event. The *Effect* of the event in alignment with causation frame is the transfer of glucose through the membrane of a cell. The *Effected* entities are the cells which states are changed as an result of the event.[1]

[1]In coming chapters we would continue with the example of *"insulin forces storage of glucose in liver cells"*; but we will not identify *"liver cells"* as the *effected* entities. This is due to the fact that the valence underlying the considered interpretations identifies the subphrase *"storage of glucose in liver-cell"* as the *effect* of the causation event. Other valences underlying the semantical analysis might result in interpretation like the one above.

3.2.2 Word Meanings and Lexical-units

A *Lexical-unit* is a pairing of a word with a frame, which represents the meaning of the word. A word (a lemma in lexicography) evokes a frame if it is a lexical-unit, i.e. paired with the frame. Frame-element realizations are the syntactic dependents of the frame-evoking words. Usually the frame-evoking words are verbs of a and FEs realizations are syntactic dependents of the verbs. Syntactic dependent are complements and specifiers (see **??**).

A word essentially evokes a frame if the meaning of the word is assumed to be captured by the frame. Below two sentences - (3.5) and (3.6), describing the same event, namely the causation-event depicted by Figure 3.1 are given; a similar causation-event within the same context of cellular actions of insulin is given as well(3.7).[2]

Insulin **forces** storage of glucose in liver cells. (3.5)

Insulin-stimulus response **causes** storage of glucose. (3.6)

Reduction of glucose-content in blood is **brought about** by insulin-stimulus response. (3.7)

A word can have different lexical-units with respect to different meanings of the word. Different lexical-units have different syntactical and correspondingly semantical patterns and combinatory possibilities. Considering a sentence governed by a frame-evoking word (in FrameNet jargon referred to as the *Target*) makes the word (as mentioned before) a predicator that takes the word's syntactical dependents as arguments. E.g with respect to (3.5) *force*, is the predicator that determines how the syntactical dependents are semantically interpreted. I.e. *insulin* as the Actor, *"storage of glucose"* as the Effect and *liver-cells* as the Effected entities, in accordance with causation frame that is evoked by *force.v*.[3]

[2]The sentence(3.5) is an example from a Wikipedia-article on Insulin [6]; (3.6) and (3.7) are constructed sentences based on valence-patterns of *cause.v* and *bring_about.v* lexical-units of causation-frame.[4]

[3]The verb *force* is actually not a lexical-unit of causation-frame in FrameNet; the assumption made here that it is a lexical-unit, is supported be lexical-data from VerbNet and WordNet. These are as follows:

VerbNet: Verb-class force-59 some members: coerce,pressure,induce,lead

WordNet-2.1: force.v Sense 1: coerce,pressure,force (cause to do through pressure of necessity)
Due to the hypernymy which is generalization/Is_A -relation in WordNet, the following synonym-set is the hypernym of the sense-1 of force.v induce, stimulate, cause, have, get, make (cause to do; cause to act in a certain manner)

SR-layer	Actor	target	Effect	Effected
PT-layer	NP	V[force]	NP	PP[in]
ST-layer	+substance -glucose	*none*	+process -regulation	+substance -insulin

Table 3.3: A Valence-pattern of the LU *force.v*

3.2.3 Frame Valence

Each lexical-unit has different syntactic realizations of its frame-element. Valence is the relation between acceptable syntactic patterns headed or governed by a word, and the corresponding combination of semantic-roles. Valence basically determines the semantic-roles of each dependent. Different syntactic patterns has different meanings.

FrameNet project is an attempt of giving an comprehensive account of valence patterns of words in English. Valence in terms of syntactical co-occurrence possibilities of words were introduced in previous charter; but valence is closely related to semantics, and semantics of a phrase is in return determined by the syntactical structure comprising the phrase's valence.[10] I.e. semantic of a phrase is the semantical structure with constituents corresponding to syntactic dependents of the phrase's head-word.

The reason to use frames is, because generative grammars of English, does not provide the relation between valid syntactic patterns and the corresponding semantic realizations, with respect to a specific underlying meaning (frame semantics). Syntactic and semantic combinatory possibilities for each lexical-unit provide a comprehensive mapping from form to meaning.

The basic aim with FrameNet is to account for the range of all acceptable syntactic constructions with respect to an intended meaning (frame) and a target-word (frame-evoking word). Not all syntactical valid constructions are allowable in English depending on the target-word. But an even more convincing reason for considering frames as a means of natural-language processing and semantic interpretation, is that not all allowable syntactical constructions can be realizations of acceptable (or desirable) semantic-interpretation.

Valence patterns are essentially the rules that constitute the grammar of our semantic parser. In order to understand valence better an example is consider here; and the structure of valence-patterns is explained with help of the valence-pattern illustrated here. Table 3.2.3 shows one valence-pattern of the LU *force.v* with respect to the causation meaning of the word. Valence patterns are structures in four-layer. The first layer is the *semantic-role* layer (SR-layer) which

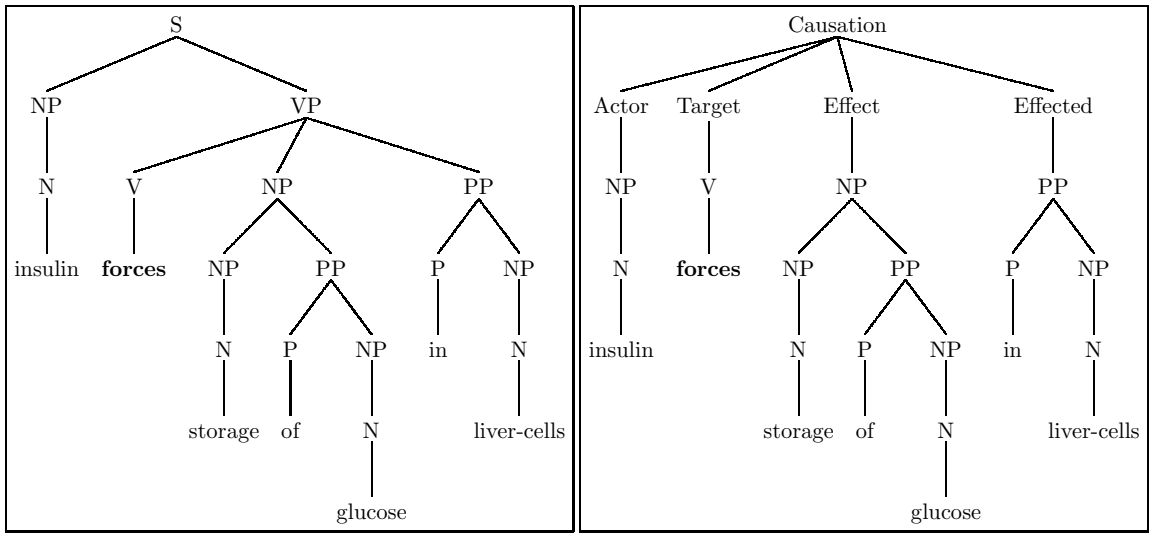


Figure 3.2: English and Frame ParseTrees of sentence (3.5)

consists of a combination of some the frame-elements and the target-word. The second layer is the *phrase-type* layer (PT-layer) which is a combination of phrase-types (phrasal categories); these are the dependents (complements and specifier) of the target word and are the syntactical realizations of the frame-elements. The third layer is the *grammatical-function* layer which is the pattern consisting of the grammatical-functions such as object, subject etc. In domain specific (or deep semantic-interpretation) a fourth layer which is *semantic-type* layer (ST-layer) is very important. The semantic-type layer consists of the ontological-types (concepts) of the given constituents. The grammatical-function layer is of no interest, since it does not contribute significantly to the semantics. Therefore it is omitted in Table .

3.2 the shows the ordinary, and the corresponding frame-based parsing of the sentence *"insulin forces storage of glucose in liver-cells"*. This illustrates how parsing based on grammars consisting of valence-patterns, can directly yield semantic annotation of the parsed text.

3.2.4 Semantic Constraints

While the first two layers (semantic-role and phrase-type layers) are part of the parse-trees, the semantic-type layer is part of semantic evaluation/verification.

I.e. any realizations of frame represented by the valence should satisfy all the semantic constraints of this layer.

As we will see these semantic constraints are to be imposed, by means of formal ontologies in order to provide deep semantic evaluation. Above in Table 3.2.3 the semantic constraints, constituted the semantic-type layer of the valence-patterns. Of particular interest is what type a frame-element filler can be and what it can not be.[4]

3.3 Frame Networks and Relations

Frame-to-frame relations are introduced in order to express relations among frames. Frames are related in several way. The most important relations are relations considering generalization (*inheritance*- and *uses*- relations).

3.3.1 Frame-to-Frame Relations

There are three important frame-to-frame relation that are briefly described below.

Inheritance: A frame A fully inherits frame B if B has all FEs of A. FEs of B may have different names and semantic-types that are derived (subtypes) from FEs of A.[2]

Uses: A frame uses another frame when not all frame inheritance-relations of the frame-elements are made explicit (not-full inheritance). The relation expresses generalization similar to inheritance.[2]

Subframe: A frame A is a subframe of frame B, when B represent a compound event consisting of more than one event, one of which is represented by A.[2]

When for each FE in a parent-frame (inherited frame) there is a corresponding FE in the child-frame (the inheriting frame) there is a *full-inheritance* relation among the frames.[2]

Figure 3.3 shows the relations between *Eventive-affecting*-frame and *Causation*-frame and *transfer*. While the relation among eventive-affecting and causation

[4]This is in accordance with how semantic constraints are defined in VerbNet.

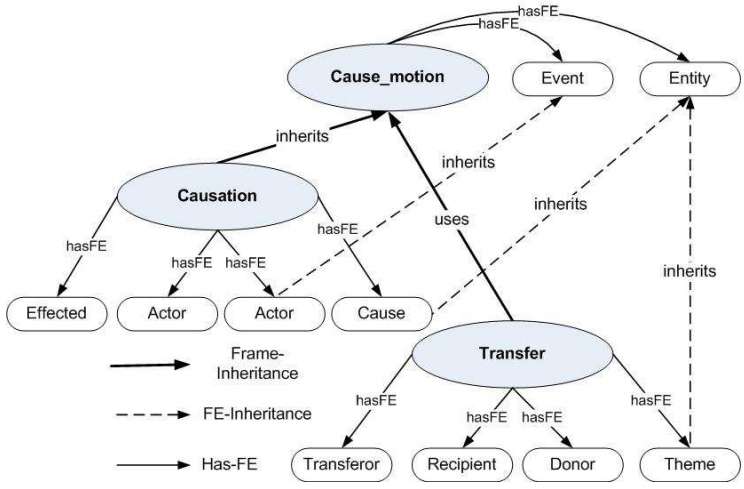


Figure 3.3: Eventive-affecting Inherited by Causation, Used by Transfer

is that of full-inheritance, transfer only uses eventive-affecting to indicate the mere generalization-relation among these two.

3.3.2 Frame-Element Relations

As briefly explain in 3.3.1 frame-inheritance requires that FEs of the inheriting-frame have either same semantic-type as the inherited-frame or the types of the FEs (of inheriting-frame) are subtypes of the inherited-frame. When this is the case the relation among the FEs are *monotonic*.[2]

We are not going to use frame-relation for automated reasoning of any form; but being aware of relations among frame helps when the right specializations of one frame are chosen to represent the semantics of a concept in more specialized way, then it would be with the super-frame. (See 5.3.3 for an example on this)

3.4 FrameNet

Purpose of Berkeley FrameNet project is essentially to give an account of valence-patterns for each lexical-units,that are identified in English. This is not an complete account firstly because the project is not completed, secondly because only

recognized lexical-units within the used collection of text are accounted for.

The starting point for identification of each lexical-unit is a frame. This means that neither all lexical-units for e.g. a certain verb are identified, nor all lexical-units for a frame are accounted for. As an evidence of this there are examples of synonymous or nearly synonymous words, by other lexical-resources that are not recognized in FrameNet (e.g. cause and force). While the verb *force* is paired with *Cause-Motion*-frame it is not paired with the more general *Causation*-frame.

Chapter 4

Parsing

This chapter is a tutorial on parsing and issues related to parsing. Different strategies and the related algorithms, that make use of these strategies will be studied. Prolog implementations of some of these algorithms is included to help the discussion since the rigorous semantics of Prolog, is assumed to be clear for the reader.

The relevance of the algorithms studied here must not be regarded restrictedly with respect to the actual implementation. The implementation technique chosen for the system is a DCG-based implementation, that uses the tabling-feature of XSB-Prolog. This makes the underlying parsing technique similar to an active chart parser, a variant of Earley's algorithm.[14] The study of parsing techniques documented in this chapter, is a reflection of the process the project went through before the choice of technology was made. This should be regarded as an attempt to justify the choice. The hope is to establish the ground for an appreciation of the technology (tabled DCG) used for the implementation.

First in Section 4.1 we will look at different dimensions and principles in parsing. In Section 4.2 we will be considering principles of bottom-up parsing; and in Section 4.3 we will look at recursive descendent (top-down) parsing. In Section 4.4 left-corner parsing is considered. Section 4.5 explains the principles of active-chart parsing; and in Section 4.6 we look at Earley's parsing algorithm which is an active-chart parser variant. Finally at Section 4.7 we will be ex-

plaining the technology used for the implementation; we show how DCG-based systems can be extended with attributes in order to evaluate semantics.

4.1 Parsing Strategies

With respect to the aim of implementing a domain specific semantic parser, there are some eminent issues to be addressed with relation to the different ways parsing is conducted. Concerning the bigger problem (bigger than pure parsing issues) with regard to the choice of technique for the implementation, it must be considered here that not only efficiency and left-recursion (typical parsing issues) are important, but also having a natural way (notation) of expressing semantics of parse-trees with regard to attribution of grammatical categories is desirable as well. The latter is with regard to difficulties of e.g. implementing an attribute extended parser.

To be able to understand difficulties pertaining to the implementation of a semantic parser (issues concerning parsing as well as semantic evaluation), it is preferred to start with the three main dimensions (or principles) in parsing, which stipulate the basic conditions for any parsing technique. Explanations of the different techniques, differing from each other with respect to these dimensions, and principles they uphold, are given afterwards in the following sections.

4.1.1 Dimensions and Principles in Parsing

The core task of parsing (recognition) is to determine whether a string of words is a well-formed sentence of a language or not. In order to accomplish this task several parsing strategies can be considered. Parsing strategies basically evolve around three dimensions. The parsing strategies and the corresponding algorithms that adhere to the principles of these strategies are differentiated with regard to the position they take on these three dimensions. These dimensions are shortly introduced here, but are comprehensively illustrated through the introductions of the different approaches to parsing in the following sections.

Goal or Data driven: this dimension concerns with the direction of the parsing process. The goal-driven approach is taken when wellformed-ness of an input string is determined by setting the goal of deriving the input string from the grammar rules. Starting off with on of the production rules of the start-symbol S, the attempt is to recursively derive matching sub-strings

of the constituents (of S) that combined together in accordance to the production rule- corresponding Right-Hand Side(RHS)- will match the input string. The data-driven principle is pursued when wellformed-ness is determined by incrementally building constituents. The process starts with the words in the input string and ends with a combination of constituents that matches at least one of the structural patterns of start-symbol(one of the RHSs of S).

Derivation or Phrase-buildup Direction: this dimension reflects the way production rules are used. More precisely this dimension concerns, in what direction strings of symbols (terminals and non-terminals) are matched against rules. I.e. whether based on the RHS of a rule - it can be based on the left-most constituent or whole of the RHS - the Left-Hand Side(LHS) is determined, or whether by selecting the LHS of a rule a commitment to the corresponding constituents is made.

Handling Non-determinism: this dimension is concerned with the way derivation or matching is handled with regard to possible alternatives. This dimension essentially concerns whether only one matching or derivation is pursued all the way until it fails (or succeeds), or several possible matches and derivations are pursued simultaneously. The first approach results in a *depth-first* search that e.g. - in the case of top-down parsing - recursively drives one constituent at a time, by trying one production rule at a time. The latter approach results in a *breath-first* that at each step can pursue all the different alternatives simultaneously.

4.2 Bottom-up Parsing

Generally when a data-driven or bottom-up approach is pursued, constituents are built starting at the level of words. Matching the words against RHSs of grammar rules lexical-categories are determined their; and these will incrementally be combine into bigger constituents. The process is essentially a series of steps, and at each step results from previous steps are combined to build bigger categories. At each step, if the pattern of a sub-string of the symbols at hand matches the pattern of a rule's RHS, the parsing process will arrive at the next state where the newly recognized LHS symbol has replaced the matching substring. This newly recognized symbol will respectively combine with other symbols to constitute even bigger constituents in coming steps. This incremental approach will finally lead to the built-up of the start-symbol's constituents, and at the very end to the built-up of the start-symbol itself. Further it is of interest here - with regard to phrase-buildup direction consider above - to emphasize the way rules are used. Rules are used from right-to-left.

In order to illustrate bottom-up parsing consider the grammar rules in Table 4.1 that stipulate some of the acceptable storing-frame realizations. The input string of *"storage of glucose"*, can be parsed using the rules in Table 4.1. This string can be parsed in a bottom-up fashion, starting from left in the input string and matching the words with some of the rules' RHSs to determine their lexical-categories. Combinations of recognized categories ordered in accordance to some rule can constitute bigger constituent, that can in return form ⟨*noun(storage)*⟩⟨*theme*⟩, forms ⟨*storing*⟩.

This process is illustrated by Table 4.2, that lists the sequence of steps taken by the parser. The parsing process is illustrated in the table by showing the rule used at each step, and the state of parsing in terms of a string of symbols that consists of the recognized categories and terminals (words) yet to be processed. Parsing this particular input string can be summarized as follows: processing the words in the input string *"storage of glucose"* from left to right, *storage* is recognized (steps 1-2) to be a ⟨*np(storage)*⟩. *of glucose* is recognized to be ⟨*pp(of)*⟩ (steps 3-6), which is in return is a ⟨*theme*⟩ (step 7). At the end we have a string of non-terminals ⟨*np(storage)*⟩⟨*theme*⟩ that matches the one of the RHSs the ⟨*storing*⟩ namely ⟨*np(storage)*⟩⟨*theme*⟩.

The process just described, corresponds to going from the bottom of the parsing tree in Figure 4.1 to the top, starting from left. Therefore this approach is a bottom-up and depth-first parsing strategy. It is a depth-first search because constituents are built as big as they can get from left, before remains of the string are processed.

Before ending this section it is in place to consider the following facts about the grammar rules in Table 4.1. The categories with arguments e.g. ⟨*np(storage)*⟩

$$
\begin{aligned}
\langle storing\rangle &\rightarrow \langle np(storage)\rangle\langle theme\rangle\langle goal\rangle\\
\langle storing\rangle &\rightarrow \langle np(storage)\rangle\langle theme\rangle\\
\langle theme\rangle &\rightarrow \langle pp(of)\rangle\\
\langle goal\rangle &\rightarrow \langle pp(in)\rangle\\
\langle np\rangle &\rightarrow \langle noun\rangle\\
\langle np(X)\rangle &\rightarrow \langle noun(X)\rangle\\
\langle pp\rangle &\rightarrow \langle prep\rangle\ \langle np\rangle\\
\langle pp(X)\rangle &\rightarrow \langle prep(X)\rangle\ \langle np\rangle\\
\langle noun(storage)\rangle &\rightarrow \text{storage}\\
\langle noun\rangle &\rightarrow \text{glucose}\\
\langle prep(of)\rangle &\rightarrow \text{of}\\
\langle prep(in)\rangle &\rightarrow \text{in}
\end{aligned}
$$

Table 4.1: Storing-Frame Grammar Rules

Step	Production Rule	Parsing State
1	$\langle noun(storage)\rangle \rightarrow$ storage	$\langle noun(storage)\rangle$ of glucose
2	$\langle np(storage)\rangle \rightarrow \langle noun(storage)\rangle$	$\langle np(storage)\rangle$ of glucose
3	$\langle prep(of)\rangle \rightarrow$ of	$\langle np(storage)\rangle\langle prep(of)\rangle$ glucose
4	$\langle noun\rangle \rightarrow$ glucose	$\langle np(storage)\rangle\langle prep(of)\rangle\langle noun\rangle$
5	$\langle np\rangle \rightarrow \langle noun\rangle$	$\langle np(storage)\rangle\langle prep(of)\rangle\langle np\rangle$
6	$\langle pp(of)\rangle \rightarrow \langle prep(of)\rangle\langle np\rangle$	$\langle np(storage)\rangle\langle pp(of)\rangle$
7	$\langle theme\rangle \rightarrow \langle pp(of)\rangle$	$\langle np(storage)\rangle\langle theme\rangle$
8	$\langle storing\rangle \rightarrow \langle np(storage)\rangle\langle theme\rangle$	$\langle storing\rangle$

Table 4.2: Bottom-up Parsing of *"storage of glucose"*

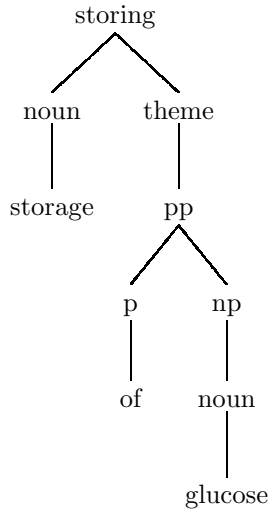


Figure 4.1: Parse-tree of *"storage of glucose"*

and ⟨*pp(of)*⟩ emphasize the acceptable words for the categories. E.g. ⟨*np(storage)*⟩ means that the noun-phrase must be with the word *storage* - which is the target of the storing frame. Similarly ⟨*pp(of)*⟩ stipulates, that the acceptable preposition is *of* - this is due to the valence-patterns of the target *storage*.

4.3 Top-down Depth-first Parsing

Top-down parsing can be breath-first or depth-first. In the following the depth-first approach is illustrated and explain. The principle of breath-first strategy is illustrated in 4.5.1, where a breath-first algorithm is studied.

A goal-driven or top-down parsing strategy starts out with the goal of deriving sentences that match the input string, i.e. the start-symbol of the language's grammar is the main goal. The input string should at least satisfy one of the S's RHSs, i.e. the input string must be derivable from at least one of the S's alternatives. Matching the data against one RHS of S means to replace the main goal, with the sub-goals of deriving the corresponding constituents.

In depth-first search whenever there is more than one choice for a category, i.e. more than one production rule representing the same non-terminal, the search will go on with the first alternative until it fails. If it succeeds with a sub-goal it will keep the result and process the symbol-string at hand based on the results so far. If it fails it will reconsider the latest sub-goal by backtracking and trying with the sub-goal's other alternatives. When the grammar is ambiguous, that is it provides more than one parsing tree for an input string, it corresponds to all possible backtrackings in a derivation tree that leads to a successful parse.

The search tree (derivation-tree) in Figure 4.2 illustrates top-down parsing of the string *"storage of glucose"* with respect to the rules in Table 4.1. The tree mimics the search a standard Prolog system (SWI-Prolog) goes through for parsing the input string, provided with the grammar in Table 4.1 (e.g. in DCG-based parsing).

The process works as follows: starting with the sub-goals at each step from left to right, the derivation will go on with the left-most sub-goal until it is resolved - it is matched at word level - or it fails. The later causes the system to backtrack to the node above to try to resolve the failing sub-goal with alternative rules. When all subgoals are resolved, the input string is recognized and is derivable from the main-goal that was the start-symbol. Considering the derivation-tree above first branch in the tree - with node (2) as top - fails, since the derivation does not match the empty string. The second branch is initiated when the system

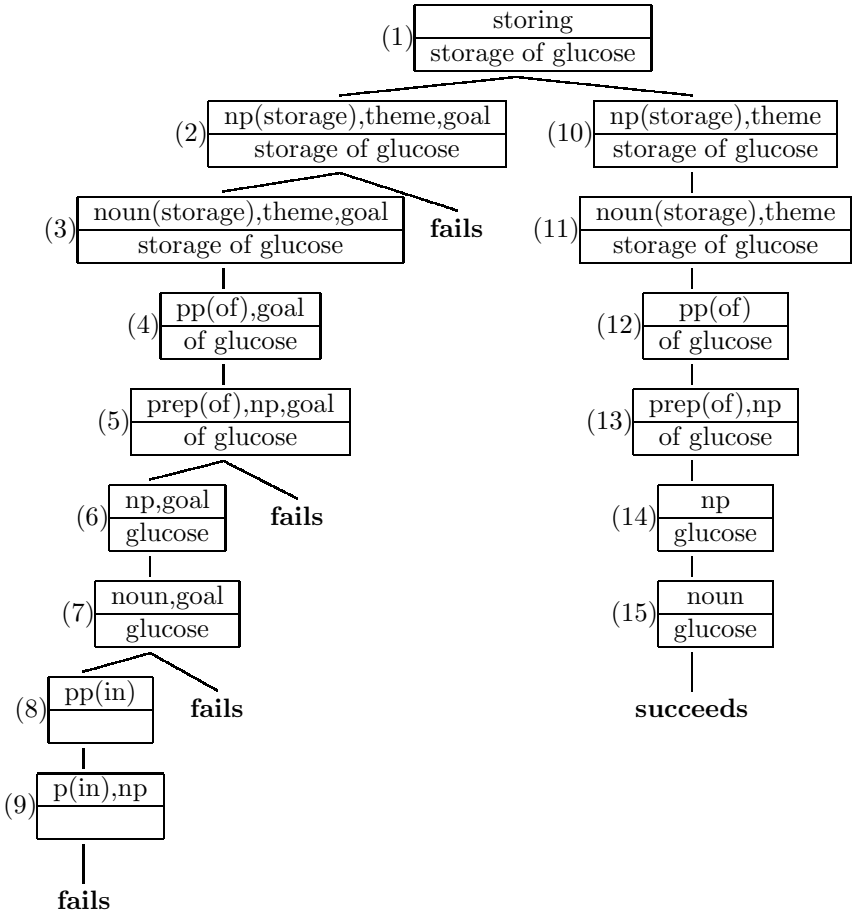


Figure 4.2: Top-down Depth-first Parsing

backtracks all the way back to the top of the tree, and begins the derivation of the second alternative of the start-symbol.

A crucial issue to be observed here about top-down parsing is that, when the system backtracks the derivations that already are recognized to match the string are done again. I.e. top-down parsing or recursive-descendent parsing - as it is also called - inherently has some efficiency issues that are related to redundant repetitions. The same point can be made about bottom-up depth-first parsing, since some alternatives may backtrack and new matching must be made to build bigger constituents at higher levels.

4.4 Left-Corner Parsing

Left-corner parsing takes advantage of combining top-down and bottom-up techniques. The essential idea of left-corner parsing is to combine top-down predictions with bottom-up built-up of constituents in order to avoid weaknesses of pure top-down or bottom-up techniques. Left-corner parsing - though interesting in itself - is not efficient enough to be considered for implementation. But introducing left-corner algorithm here helps us to understand the underlying principles of an algorithm with more desirable characteristics, namely Earley's algorithm.

In the following the motivation for combining top-down and bottom-up approaches in order to get a more efficient parsing is considered; and left-corner parsing itself is introduced after wards.

4.4.1 Motivation

The motivation for left-corner parsing is to overcome inefficiencies inherently present in top-down and bottom-up approaches. When top-down parsing is conducted, the process starts at each step with some prediction/goal (e.g. $\langle np \rangle$) and uses the RHSs of the corresponding production rules one at a time to produce the pertaining derivations. When each derivation is at a level where no other rules can be applied the input string is matched against RHS of the last descendent (e.g. $\langle noun \rangle$ is last descendent of $\langle np \rangle$). If no match is found the process will backtrack and choose an alternative RHS of the prediction. The main point is that alternatives of LHS are chosen with no regard of how the structure of the input string looks like. This means that the process in worst case must backtrack lots of times, where derivation of descendants are repeated.

In order to illustrate this consider the following the grammar - that is the simplified version of the grammar in Table 4.1 - below. The input string is *"storage of glucose"* again. The parser is deriving the first alternative of $\langle storing \rangle$ - corresponding to rule 4.1) -, and is at a stage where it has recognized the three words in the input string to have the following structure $\langle np \rangle \langle theme \rangle$ and is left with the empty string. The parsing will proceed looking for $\langle goal \rangle$, which results in backtracks starting at the level of the rules (4.8) and(4.9). This is because none of RHSs of $\langle prep \rangle$ match the empty string. Since all other descendants backtrack too, the process will backtrack all the way to the top - the main goal $\langle storing \rangle$. The parser will proceed at this stage by attempting to match the input string with an alternative derivations of $\langle storing \rangle$, using the rule (4.2). It derives the structure $\langle np \rangle \langle theme \rangle$ again, which is redundant since this is done once already. The tree in Figure 4.2 shows this process in details.

$$\langle storing \rangle \rightarrow \langle np \rangle \langle theme \rangle \langle goal \rangle \tag{4.1}$$
$$\langle storing \rangle \rightarrow \langle np \rangle \langle theme \rangle \tag{4.2}$$
$$\langle theme \rangle \rightarrow \langle pp \rangle \tag{4.3}$$
$$\langle goal \rangle \rightarrow \langle pp \rangle \tag{4.4}$$
$$\langle np \rangle \rightarrow \langle noun \rangle \tag{4.5}$$
$$\langle noun \rangle \rightarrow glucose \tag{4.6}$$
$$\langle noun \rangle \rightarrow storage \tag{4.7}$$
$$\langle prep \rangle \rightarrow of \tag{4.8}$$
$$\langle prep \rangle \rightarrow in \tag{4.9}$$

In the bottom-up approach the process starts at the bottom with the input string. The aim is to combine input words to build constituents, and combine these constituents to bigger constituents, all the way to the start-symbol. In this process any constituents that can be built using the grammar rules from right to left are built; with no regard to whether it will match a well-formed phrasal-structure further up in the process. The process backtracks if no RHS is matched. This can be at the cost of a series of repetitions, where alternative built-ups are considered. This is in particular the case when the grammar is ambiguous.[12]

4.4.2 Left-corner Algorithm

In what follows the left-corner parsing algorithm is introduced. The introduction will be started by defining the left-corner of a production rule, as the first (left

most) symbol on the RHS of a rule. E.g. $\langle np \rangle$ is the left corner of the rule $\langle storing \rangle \rightarrow np\langle theme \rangle\langle goal \rangle$, similarly *glucose* is the left corner of the rule $\langle noun \rangle \rightarrow glucose$.

There are three different operations that are of interest with regard to left-corner parsing here, and *active chart parsing* algorithms introduced in following sections; these operations are *prediction*, *scanning* and *completion*. But before considering these operation the notion of a *complete category* should be considered, because of the central role it plays. A category is complete if its considered RHS - the RHS of the rule under consideration - is derived, i.e. all of its constituents are completely derived. Having introduced this notion, the underlying ideas of prediction, scanning and completion are introduced below. The aim is to indicate the idea of these operations, as the concrete implementations of these operations vary from algorithm to algorithm.

Prediction: Prediction is the top-down operation of restricting the parsing process to a goal, that is the category to be derived. Left-corner parsing starts by making the prediction, that the input string will make a sentence, i.e. the starting goal will naturally be the start-symbol - similar to top-down parsing. In left-corner parsing as the process proceeds and categories are completed bottom-up, new predictions are made to guide the parsing process.

Scanning: Left-corner parsing proceeds by determining the category of the word ahead in the input string, and trying to complete the current goal in a recursive process. Scanning is the simple operation of using words or other complete categories to advance the completion of the current goal. Scanning is generally part of the a bottom-up approach that consist of completing categories starting from the lexical-categories of the word ahead; and then making new predictions based on the rules that the completed category at hand is the left-corner of.

Completing: Completion is the operation that combines the bottom-up and top-down principles. Completion essentially makes use of scanning and prediction to complete the current goal's constituents from left-to-right. Scanning as it is described above starts off the recursive bottom-up completion of the sub-categories from left to right guided by the new predictions.

As indicated above these operations are mutually dependent (mutually recursive). In order to summarize what was implicit above; left-corner parser basically alternates between bottom-up scanning and top-down predictions to complete categories; in the completion process new predictions are made as old predictions are completed recursively from left to right, starting with the lexical-category of the next word in the input string.

```
1  leftcorner_recognizer(StartSymbol, Wordlist):-
2          scan(StartSymbol, Wordlist, []).

4  scan(Prediction, [Word| Wordlist], RmWordlist):-
5          lex(Word, LexCat),
6          complete(Prediction, LexCat, Wordlist, RmWordlist).

8  complete(Prediction, Prediction, Wordlist, Wordlist).
9  complete(Prediction, CompleteCat, Wordlist, RmWordlist) :-
10         LHS ---> [CompleteCat| Predictions],
11         predict(Predictions, Wordlist, RmWordlist1),
12         complete(Prediction, LHS, RmWordlist1, RmWordlist).

14 predict([], Wordlist, Wordlist).
15 predict([Prediction| Predictions], Wordlist, RmWordlist):-
16         scan(Prediction, Wordlist, RmWordlist1),
17         predict(Predictions, RmWordlist1, RmWordlist).
```

Listing 4.1: Left–corner Recognizer

In order to make this more concrete consider the following small example. We assume that a ⟨*np(storage)*⟩ has been recognized; i.e. the first word *"storage'* of the input string *"storage of glucose"* is recognized to be and ⟨*np(storage)*⟩. The left-corner parser will proceed looking for a rule in the grammar with ⟨*np(storage)*⟩ as its left-corner. If the input string is supposed to be of category ⟨*storing*⟩, that is derivable from the ⟨*storing*⟩ → ⟨*np*⟩⟨*theme*⟩, the remaining of the input string (*"of glucose"*) has to be recognizable as a ⟨*theme*⟩. ⟨*theme*⟩ is hence the prediction, the goal to be followed unless its fails, in which case it causes the system to backtrack as usual. The left-corner parser will continue alternating between bottom-up and top-down steps as described above until it has recognized ⟨*theme*⟩, thereby completing the sentence.

Listing 4.1 shows the Prolog implementation of left-corner recognizer. [1] First there is the main prediction (lines 1- 2) that the input string(*Wordlist*) is a derivation of *StartSymbol*. The parsing process starts off by scanning the first word of the input string, and trying to complete the main prediction recursively in a bottom-up manner by means of the recursive predicate *complete* (lines 9-12). If the scanned *LexCat* or completed category *CompleteCat* is the left-corner of a rule (line 10) the process proceeds by predicting by means of *predict* predicate all the remaining constituents of the rule (*Predictions*). If the predictions succeed,

[1] The source of this program is [12]. In ordet to enhance the understanding of the algorithm the predicates are renamed to adhere to the terminology used here - in particular the operations considered above, *scan*, *complete* and *predict*). Furthermore reader friendly argument names have replaced the original ones.

```
1  storing ---> [np, theme].
2  theme ---> [pp].
3  np ---> [noun].
4  pp ---> [prep, np].
5  lex(storage, noun).  lex(glucose, noun).
6  lex(of, prep).

8  link(np, storing).
9  link(pp, theme).
10 link(noun, np).  link(noun, storing).
11 link(prep, pp).  link(prep, theme).
12 link(X,X).
```

Listing 4.2: Tiny Frame–based Grammar with Links

that is *LHS* has completed, the attempt of completing the current prediction (*Prediction* in line 9) continues recursively until the process arrives at the current prediction (in line 8) as the latest completed category. The role of *predict* predicate (lines 15-17) is to set the next constituent - after left-corner - as the new goal, and continue this until all constituents are completed(line 14).

The left-corner parsing is illustrated in Figure 4.3 where the combined top-down prediction and bottom-up completion is depicted by showing how the parsing tree is gradually built. The example below (Figure 4.3) shows left-corner parsing of our input string, based on the grammar listed in Listing 4.2. Figure 4.3 shows e.g. that at step (2) an $\langle np \rangle$ is recognized (completed) and at step (3) $\langle theme \rangle$ is predicted in accordance with left-corner algorithm, since $\langle theme \rangle$ is the second constituent of $\langle storing \rangle$.

Here follows some facts about left-corner parsers and the way left-corner parsing is introduced here. Left-corner parsing is introduced here as a front runner for the more complex but efficient algorithm of Earley's parsing algorithm, that like left-corner algorithm combines a bottom-up approach with top-down predictions. The operations of *scan*,*complete* and *predict* introduced above are somehow imposed here, since the operations are mutually recursive and don't adhere to pure scanning, prediction and completion in the way these three operations are defined and used in Earley's algorithm (See Section 4.6). But due to the introduction of the Earley algorithm later on in this chapter, it is preferable to introduce the concepts of scanning, prediction and completion here.

One of the advantages of left-corner parser is that it does not have a left-recursion problem. Left-recursion occurs when a recursive rule has itself as the left-corner. In the top-down (or recursive-descendent) approach this will

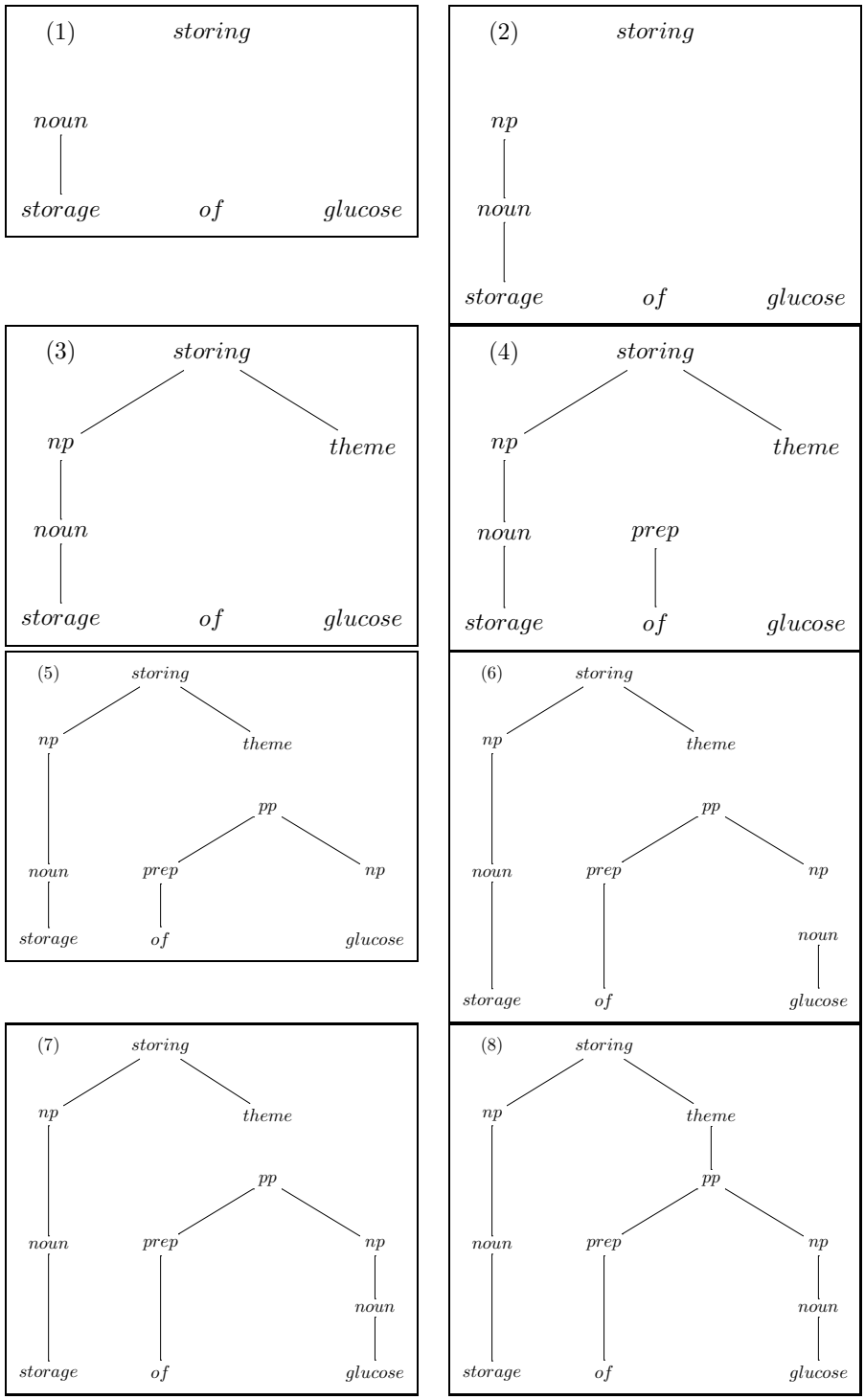


Figure 4.3: Illustration of Left-corner Recognition

lead to an loop where the sub-goal is the goal itself all the time. But since in a bottom-up approachs the rules are matched from right to left, i.e. first the constituents are built and then the corresponding category the process will not end up in a loop.

One of the weaknesses of left-corner parser - as it is introduced in Listing 4.1 - is that, if the grammar is ambiguous left-corner parser suffers from a similar problem as pure bottom-up parser, namely completed categories that can not constitute the prediction. There must be a means of determining that a category will eventually lead to the left-corner of the current prediction, and others will not. This information can be provided by links, that link any *Category1* with any other *Category2* whose left-corner is *Category1*; or *Category1* is the left-corner of a category that is linked to *Category2*. Completion of a category that is not leading to the prediction can be stopped using the information provided by links. I.e. the completion of a category, which is not linked to the prediction, will not be initiated. The tiny grammar and the corresponding links for the considered example (Figure 4.3) are listed in Listing 4.2.

4.5 Active Chart Parsing

In this section *active chart parsing* will be introduced. The study of parsing strategies and algorithms has now arrived closer to its destination, Earley's algorithm which is an active chart parser.

The algorithms we have considered so far, do not use any external data-repository. I.e. they do not use any data-structure for the purpose to store intermediate results and auxiliary information. All of this kind of intermediate data are implicit to the parsers. Chart parsers on the other hand (active as well as passive) use data-structures to store information about the state of the parsing process.

A chart in this context can be described as an account of the information available to the parser during the parsing of a particular string. These information are basically at to levels, at the level of words and at the level of the information pertaining to processing of categories. The input string which is essentially an ordered sequence of words, - ordered with respect to the words positions - is represented by a data-structure that conveys the positional information of each word in the input string as well as the word itself. Further data-structures to represent the state of the parsing process in terms of categories processed or under processing are used. These structures represent the so called *dotted rules* accompanied with positional information to give an account of the progress of each category in the process.

In dotted rule notation, the derivation state of category X given the production rule $X \rightarrow \alpha\beta$, where α is recognized and β is expected, can be represented as the following $X \rightarrow \alpha \centerdot \beta$. The dot in dotted rule notation of this form, indicates how far derivation of the category X has progressed, in terms of the constituents of the category, with respect to the particular rule ($X \rightarrow \alpha\beta$). Dotted rule notation combined with positional information is a way of representing where in the input string, search for a category using one of its production rules, started and how far it has progressed. In particular these are structures of e.g. this form $(\gamma \rightarrow \langle np \rangle \centerdot \langle theme \rangle, 0, 1)$, which essentially represent the state of the category in terms of completion.

A snap shot of the parsing process of the input string *"storage of glucose"* is represented below in terms of derivation state notations (just introduced). The first word is recognized to be a $\langle np \rangle$. (4.12) represent this state, i.e. derivation of $\langle np \rangle$ have been started at position 0, and after recognition of the first word as a $\langle noun \rangle$ - represented by 4.13 - $\langle np \rangle$ has completed at position 1. (4.10) represent the state of deriving $\langle storing \rangle$, that started at position 0 and is at position 1 after completing the first constituent ($\langle np \rangle$) and is looking for a $\langle theme \rangle$ now. (4.11) represent the state of $\langle theme \rangle$ which is predicted and deriving it has just started at position 1.

$$(\langle storing \rangle \rightarrow \langle np \rangle \centerdot \langle theme \rangle, 0, 1) \tag{4.10}$$
$$(\langle theme \rangle \rightarrow \centerdot \langle pp \rangle, 1, 1) \tag{4.11}$$
$$(\langle np \rangle \rightarrow \langle noun \rangle \centerdot, 0, 1) \tag{4.12}$$
$$(\langle noun \rangle \rightarrow storage \centerdot, 0, 1) \tag{4.13}$$

The states above joined together represent the chart in Figure 4.4. The chart is the words positioned on a line, with arcs representing the dotted production rules that go from position to position. The chart represent the state of parsing process in terms of arcs representing the derivation states of the categories.

In the following we will distinguish between active arcs and passive arcs. Arcs representing uncomplete categories are active arcs. Passive arcs represent complete categories in a chart. In the above example (4.10) and (4.11) are active arcs; and (4.12) is a passive arc. Chart parsers can use active charts or passive charts; thereby the notions *active chart parser* and *passive chart parser*. An active chart is a chart that has active arcs included as well as passive arcs - e.g. the chart in Figure 4.4. A passive chart is a one that includes passive arcs only. Passive chart parsing is when no predictions are made, i.e. no active arcs are considered and only complete arcs are built, in a bottom-up approach using the rules from right to left. A passive chart parser progressively adds more arcs to

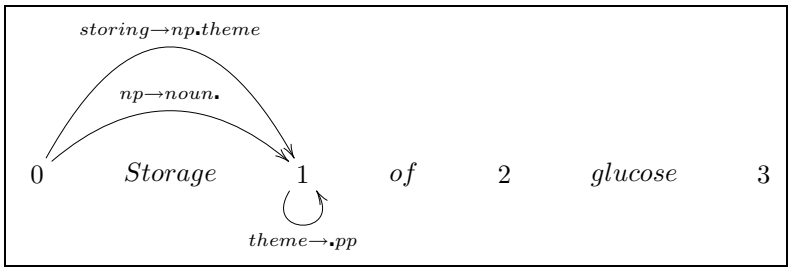


Figure 4.4: Example of a Chart corresponding to the input string "storage of glucose"

the chart as the input string is processed word by word and more complete arcs can be built right-to-left, combining complete arcs that are already included in the chart.[12] Active chart parser on the other hand function by adding predictions to the chart and completing them as the process proceeds. Active chart parsing can be top-down or bottom-up.

Active chart parsers can implement a breath-first search or a depth-first search, in case an agenda is used. An agenda is a data-structure used to keep track of newly made arcs - prediction and progressed arcs - that has to be processed. Arcs in the agenda are processed one by one, checking whether they can combine with any arc in any ways; whether they complete any arc in the chart, or they can proceed (progress) by combining with a complete constituent, that they currently expecting. Whether the agenda is a stack (FILO) or a queue (FIFO), determines the search approach. I.e. how the agenda is implemented determines whether the search is a depth-first or breath-first search. If the agenda is a stack then the search is a depth-first, as newly created arcs are pursued first. If the agenda is a queue the search is a breath-first search, because all possible choices (arcs) for e.g. a category are pursued simultaneously.[2]

The main motivation for active chart parsing algorithms - and Earley's algorithm - is to get rid of inefficiencies due to redundant repetitions. When a pure top-down or bottom-up algorithm backtracks there is a vast amount of work that is repeated. Furthermore chart parsers can handle left recursive grammars as well.

[2]Arcs predicting the category are added to the queue one after another, and will be added to the chart on after another in order to be processed.

4.5.1 Active Chart Parsing Breath-first

In what follows a general active chart parser is considered that can be implemented as a bottom-up or top-down parser, or it can be implemented to conduct a breath-first or depth-fist search. After the introduction of the general algorithm a bottom-up breath-first version of it is implemented. In Section 4.6 Earley's algorithm is introduced, which is an active chart breath-first algorithm with look ahead.

Before introducing the general algorithm consider the notion of *the fundamental rule* that plays a central role in the algorithm. The fundamental rule defines the way an active arc, can combine with a passive arc. An active arc can combine with a passive arc if the active arc expects the category of the passive arc at the position that the passive arc starts. I.e. the non-terminal at the right of the dot of the active arc, is the LHS of the passive arc as is illustrated in Figure 4.5. Figure 4.5 shows the way an active arc is combined with a passive arc right of it to create a new arc, that when added to the chart will result in chart illustrated at the bottom of the figure.

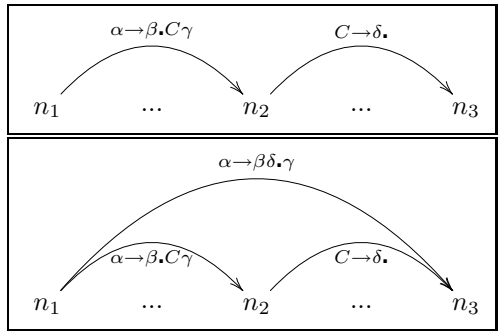


Figure 4.5: Fundamental Rule of Arc Combination

Consider the following general active chart parsing algorithm in Listing 4.3. Concrete differences, in how the steps in the general algorithm are carried out determine whether the algorithm is a top-down or bottom-up parsing algorithm.

In the bottom-up variant of the general algorithm (Listing 4.4) the chart is initialized by recording tuples that represent each word in the input string with their position (start and end positions). The chart at this initial state is equivalent to the alignment of words enclosed in positions - like e.g. in Figure4.4 without the arcs. The agenda is initialized by adding complete arcs to the agenda that corresponding to the lexical categories of the words in the input

```
1  active-chart-recognizer(input-string)
2  begin
3       initialize-chart(input-string)
4       initialize-agenda

6       while not empty(agenda)
7            do  arc←remove-first(agenda)
8                if  not is-in-chart(arc)
9                     then add-to-chart(arc)
10               complete(arc) //apply fundamental rule
11               predict(arc)

13      is-complete(start-symbol)
14 end
```

Listing 4.3: General Active Chart Parsing Algorithm

string. Since a breath-first search is aimed here, new arcs are appended at the end of a list that is the agenda, and removed from the front of the list. I.e. the agenda is essentially a queue. After the initial states the arcs are removed from the agenda and processed according to the fundamental rule and left-corner prediction(prediction as introduced in 4.4.2).

Applying the fundamental rule - Listing 4.3 line 10- means proceeding the completion of any prospective category - category corresponding to an active arc in the chart or on agenda - by combining a passive arc immediately to the right with the corresponding active arc. Prediction here is conducted like prediction in left-corner parsing (see 4.4.2). I.e. the LHS of a passive chart is matched against the left-corner of rules, if matches are found the corresponding active arcs beginning at the position where the passive arc begins and ending at the position where the passive arc ends, are added to the agenda. To illustrate this consider the complete arc (4.14), based on the rule $\langle storing \rangle \rightarrow \langle np \rangle \langle theme \rangle$ (4.15) is predicted.

$$(\langle np \rangle \rightarrow \langle noun \rangle \bullet, 0, 1) \tag{4.14}$$

$$(\langle storing \rangle \rightarrow \langle np \rangle \bullet \langle theme \rangle, 0, 1) \tag{4.15}$$

When the agenda is empty and there no more predictions to be made the chart is checked - Listing4.3 line13 - to se whether there is a complete arc corresponding to the start-symbol of the grammar, if there is then there is a complete sentence of the language that is recognized.

Listing 4.4 includes the Prolog implementation of a bottom-up, breath-first ac-

```
predict(arc(E,S,[],Cat1),Agenda,Agenda1):-
        findall(arc(E,S,RHS,Cat2),Cat2--->[Cat1|RHS],NewArcs),
        append(Agenda,NewArcs,Agenda1).
predict(_,Agenda,Agenda).

complete(arc(E1,S1,[],Cat1),Agenda,Agenda1):-
        findall(arc(E1,S2,RHS,Cat2),arc(S1,S2,[Cat1|RHS],Cat2),
            NewArcs),
        append(Agenda,NewArcs,Agenda1).
complete(_,Agenda,Agenda).

scan(arc(E1,S1,[Cat2|RHS],Cat1),Agenda,Agenda1):-
        findall(arc(E2,S1,RHS,Cat1),arc(E2,E1,[],Cat2),NewArcs)
            ,
        append(Agenda,NewArcs,Agenda1).
scan(_,Agenda,Agenda).

process_agenda([]).
process_agenda([Arc|Agenda]) :-
        add_to_chart(Arc,Agenda,NewAgenda),
        process_agenda(NewAgenda).

add_to_chart(Arc,Agenda1,Agenda4):-
       \+Arc,!,
        assertz(Arc),
        predict(Arc,Agenda1,Agenda2),
        complete(Arc,Agenda2,Agenda3),
        scan(Arc,Agenda3,Agenda4).
add_to_chart(_,Agenda,Agenda).

bottom_up_active_chart(Start,String) :-
         clean_chart,
         init_chart(String, 0),
         init_agenda(Agenda),
         process_agenda(Agenda),
         length(String, N),
         arc(N,0,[],Start).
```

Listing 4.4: Bottom–up Breath–first Implementation of The General ACP Algorithm

tive chart parser. First consider the following information about the implementation, regarding the Prolog code and design choices that are made. The predicate *findall (representation_form , condition , result_list)* is a built-in predicate that for each time a *condition* is true builds a term of the form of *representation_form* and puts it on the *result_list* . Arcs are represented here as dynamic predicates of the form *arc(End,Start,RemainingRHS,Category)*.[3]; *Start* and *End* represent the start- and end-positions of the arc; *RemainingRHS* represent the list with remaining constituents; if this list is empty the arc is passive otherwise it is active. Dynamic predicates can be inserted on demand into the Prolog-system data-base by e.g. *assertz* predicate.

With these basic facts about the implementation in place, the implementations inner workings can be studied now. Consider the *scan* operation implemented at lines 11-14. When an active arc is added to the chart and can be combined with passive arcs on the chart, each time there is such passive arc in the chart a new combined arc is created that should be added to the end of the agenda-list. I.e. *scan* is a operation that applies the fundamental rule to combine active arcs from the agenda to passive arcs in the chart, by scanning for passive arcs in the chart.

The predicate *complete* - at lines 6-9 - defines the case when a passive arc is added to the chart. In this case for each time the passive arc from the agenda can combine with a active arc in the chart, a new arc is added to the agenda. *complete* applies the fundamental rule to combine passive arcs from agenda with active arcs in the chart i.e. in the opposite direction compared with *scan*.

The predicate *predict* - at lines 1-4 - makes predictions as described above. I.e. each time a passive arc is about to be added to the chart new active arcs are created based on the rules that have the passive arc's category as their left-corner. Furthermore there is the recursive predicate of *process_agenda* - at line 16 - that essentially is the main loop of the algorithm - like the loop in general algorithm at line 6 in Listing 4.3. Finally *bottom_up_active_chart (Start_Symbol, Input_String)* - at line 29 - is the main-predicate that takes a start-symbol and an input-string as arguments, and answer whether the string is well-formed with respect to the grammar and the start-symbol.

[3]The way arcs are represented here are measured, to be more efficient - due to the inner workings of Prolog systems - then the usual representations *arc(RemainingRHS,Category, Start,End)*.[13]

```
1  predict(arc(E,_,[Cat|_],_),Agenda1,Agenda2):-
2          word(Word,E,_),
3          findall(Arc,(Cat ---> RHS,predict_aux(Cat,RHS,Word,E
               ,Arc)),NewArcs),
4          append(Agenda1,NewArcs,Agenda2).
5  predict(_,Agenda,Agenda).


8  predict_aux(Cat,[Cat1|RHS],Word,P,arc(P,P,[Cat1|RHS],Cat)):-
9          lex(Word,Cat2),
10         link(Cat2,Cat1).
```

Listing 4.5: Earley Algorithm Predict Operation

4.6 Earley's Parsing Algorithm

Earley's parsing algorithm is a variant of an active chart parser, that has the characteristics desirable for any natural language processor. As indicated before a parser that is efficient in terms of execution-time, and can handle left-recursive grammars is desirable; and Earley's parser exhibit both of these characteristics, as it is one of the most efficient algorithms - if not the most efficient algorithm - for parsing CFGs, and it handles left-recursion as well. Earley's parser is a bottom-up, breath-first active chart parser with look-ahead capabilities. In this section we will see how Earley's parser is implemented using the above implementation (4.5.1) with minor changes; and furthermore how *links* - introduced before in combination with left-corner parsing - can be used to facilitate the look-ahead capabilities of the Earley's parser.

Earley-parser's execution-time is cubic, i.e. $\Theta(n^3)$ where n is the length of the input string. It has a quadratic time $\Theta(n^2)$ for unambiguous grammars; and it handles left-recursive grammar rules as well. The Earley's parser is a bottom-up active chart algorithm; and the only difference with the bottom-up active chart parser introduced above (in 4.5.1) is a top-down prediction combined with - restricted by - look-ahead capabilities. In left-corner parsing and Earley's parsing information about how categories are *linked* together is used to discard predictions that would not drive any prefix of the remaining symbol-string at hand.

Listing 4.5 includes the only changes made to the implementation of Listing 4.4. The changes affect only the way predictions are made. *predict_aux* clause - at line 8 in the Listing 4.5 - looks at the word ahead and checks whether the candidate prediction (*Cat*) will eventually complete. As mentioned before use

of links are similar here as for the extension of left-corner parser, that used this mechanism to discard redundant predictions. I.e. categories that are not linked to the lexical-category of the word ahead in the input string will not be considered. *predict* (line 1) uses *predict_aux* to filter redundant predictions in the way just considered.

4.7 NLP using Tabled DCG

In the following section we will be considering a technique used for the actual implementation of our semantic parser. Until now in this chapter, an overview of the different parsing strategies and their strengths and weaknesses were provided. But none of these algorithms are further elaborated on in this project, in order to facilitate our semantic parser. Prolog-based technique of Definite Clause Grammars (DCG) is preferred over e.g. an extension of the Earley's algorithm considered above (Section 4.6). Though it might be argued here that DCG does not exhibit the same characteristics as Earley's parsers, we will see that some Prolog-system - XSB-Prolog and Mercury in particular - come with features (*Tabling*) that converts ordinary DCG parsing to a variant of Earley's parsing that exhibit all the advantages of Earley's parser. So at the same time as the simple and natural notation of DCG is used, the advantages of Earley's parser are facilitated by the system.

First DCG is introduced and illustrated via the familiar example of storing-frame; afterwards the notion of *Tabling* is introduced, with regard to its implementation in XSB-Prolog. At the end we consider how well-formed strings are semantically evaluated via attribution - extending grammars with attributes.

4.7.1 Definite Clause Grammar Top-down Parsing

Definite Clause Grammar (DCG) is a notation used in Prolog-systems for representation of a language. Categories in the grammar are predicates, using difference-lists. A grammar described, using predicates in this way, functions as a top-down, depth-first parser (recursive descendant), i.e. it follows the same search strategy as Prolog's top-down, depth-first deduction (SLD-resolution). This kind of top-down, depth-first derivation is considered already in Figure 4.2 that shows an dcg-based parsing of the input string *"storage of glucose"*.

Prolog-systems like SWI-Prolog and XSB-Prolog allow a DCG notation that is similar to BNF notation. A clause in DCG notation like e.g. *storing−−>np,*

```
1 storing(A,C):-np(A,B),theme(B,C).
2 theme(A,B):-pp(A,B).
3 pp(A,C):-prep(A,B),np(B,C).
4 np(A,B):-noun(A,B).
5 noun([storage|B],B).
6 noun([glucose|B],B).
7 prep([of|B],B).
```

Listing 4.6: Tiny DCG for Storing Frame

theme, which is an abbreviation for an ordinary clause, has additional implicit arguments for the involved predicates that appear at compile time. I.e. the DCG clause *storing-->np,theme* is equivalent to *storing(A,C):-np(A,B),theme(B,C)*. A grammar written in DCG-notation is compiled into Prolog predicates whit difference lists. In the following example - included in Listing 4.6 - DCG will be considered in its purest form, that is predicates with difference-lists.

Consider clause 3 in Listing 4.6 with the predicate *pp(A,C)*. The string matching the phrase *pp*, is thought to be the difference-list between the first argument and second argument. In order to illustrate DCG in action, consider the the clauses of Listing 4.6 which constitute the grammar. A derivation-tree like the one depicted in Figure 4.6 is produced by the Prolog-system, when parsing of *"storage of glucose"* is initiated by querying the system with the following goal $\leftarrow storing([storage, of, glucose], [])$. In Figure 4.6 each step in the derivation is depicted by including the rule applied in left of the arrow, and the resulting substitution (unification) at the right of each arrow.

Previously in Section 4.2 we looked at productions rules with categories that took arguments e.g. $\langle storing \rangle \rightarrow \langle np(storage) \rangle \langle theme \rangle \langle goal \rangle$ or $\langle noun(storage) \rangle \rightarrow storage$. This form of productions rules can be used in combination with frame-based grammars, since frames are target dependent. E.g. valid linguistic appearances of storing-frame can be valences of the lexical-units like *storage.n* and *srore.v*. This can be achieved in DCG by escaping the DCG-notation and using Prolog in order to restrict the category-type to a certain word. This is illustrated in Listing 4.6 below, where e.g. *noun(N)-->[N],{noun([N],[])}* restrict the derivation of the category *noun(N)* to the word *N* via the goal in the curly brackets (*noun([N],[])*). In reality the goal of e.g. $\leftarrow noun(storage, [storage|B], B)$ is replaced by the goal of $\leftarrow noun([storage|B], B)$. This mechanism of escaping DCG is more important for semantic evaluation (see 4.7.3) then for category-restriction.

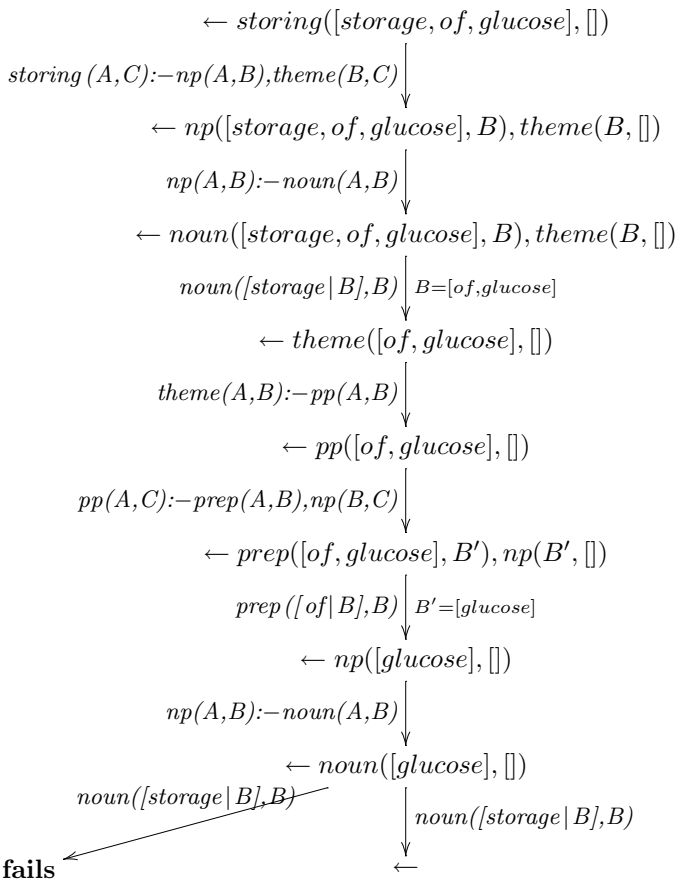


Figure 4.6: Derivation-tree of the Goal ← $storing([storage, of, glucose], [])$

```
1  storing --> np(storage),theme,goal.
2  storing --> np(storage),theme.
3  theme --> pp(of).
4  goal --> pp(in).
5  np --> noun.
6  np(N) --> noun(N).
7  noun(N) -->[N],{noun([N],[])}.
8  pp --> prep,np.
9  pp(P) --> prep(P),np.
10 prep(P) -->[P],{prep([P],[])}.
11 noun --> [storage].
12 noun --> [glucose].
13 noun --> [cells].
14 prep --> [of].
15 prep --> [in].
```

Listing 4.7: Storing–frame DCG with Category Restriction

4.7.2 Prolog Systems with Tabling

DCG is a recursive-descent parser (top-down and depth-first); hence - as considered in previous sections - it is inefficient and does not handle left-recursive grammars. These are the main motivations to look for other parsing techniques such as Earley's parsing algorithm, which is efficient and can handle left-recursive grammars. Earley's parsing algorithm is more complex, and correspondingly more difficult to implement and extend with e.g. attribution for the purpose of semantic evaluation. Fortunately there are Prolog systems like XSB and Mercury, that implement a feature that make ordinary DCG to a variant of Earley's parser. The feature is tabling, also called memoization or lemmatization.[14]

With tabling the simplicity of DCG notation remains untouched, as efficiency and left-recursion handling is provided. In the following we will use XSB-Prolog with tabling in order to implement a demo of a frame-based semantic parser. Consider the following that describes the basic idea of tabling:

"The idea is very simple: never make the same procedure call twice: the first time a call is made, remember all the answers it returns, and if it's ever made again, use those previously computed answers to satisfy the later request."[14]

The basis of tabling implementation is a forest of SLD-resolution trees that each act as a server. For each goal there is one goal-server, that derives the answers of the particular goal via SLD-resolution. Each time a tabled predicate - a

```
1 link(np, storing).
2 link(noun, np).
3 link(srorage, noun). link(glucose, noun).
4 link(pp, theme).
5 link(prep, pp). link(of, prep).
6 link(X,X).

8 linked(Cat1, Cat2):- link(Cat1, Cat2).
9 linked(Cat1, Cat3):- link(Cat1, Cat2), linked(Cat2, Cat3).
```

Listing 4.8: Transitive Closure of cyclic graphs

predicate specified to be resolved by tabling - is called a separate goal-server with a corresponding derivation-tree answers the goal, which is communicated back asynchronously. If the goal server is not already created it will be created and maintained while the program executes. Answers are send to any other requesting process (tree). This model of execution eliminates duplicates of the same goal. Furthermore this means that only a finite number of calls are made even if the definite clauses are left-recursive.[14]

Typical examples that illustrate the power of tabling well, are transitive closure of cyclic graphs. In left-corner parsing as well as Earley parsing, information about how categories are linked were used to discard non-derivable predictions. Listing 4.8 includes the definition of the recursive predicate *linked* (at lines 8-9), that defines the transitive closure of *link*-relation. The links (*link* facts) included in Listing 4.8 (lines 1-5) together with non-ground *link* predicate (at line 6) present a cyclic graph depicted in Figure 4.7.

When transitive closure is defined as in Listing 4.8 and the graph is cyclic, problems occur with goals where the answers suppose to be *NO* - with e.g. the goal $\leftarrow linked(np, theme)$. Standard Prolog-systems - systems without tabling - do not reply with any answer, because they enter a loop. The SLD-resolution tree in Figure 4.8 illustrates how an infinite loop is entered when the goal is $\leftarrow linked(np, theme)$. This is due to the fact that each category is linked to itself. Obviously if the categories were not linked to themselves - i.e. if the graph were acyclic - the program would terminate with the answer "NO". But if *linked* where left-recursive the problem would not be solved by constraining the program to trees only. This brings up the real motivation for using tabled resolution systems, which are able to handle left-recursive DCGs.

It is intuitively clear, that the goal should terminate when tabling is used, since there should be a finite number of calls for a finite graph. Figure 4.9 illustrates tabling based resolution of the problem above. Each time a goal, with *linked*-

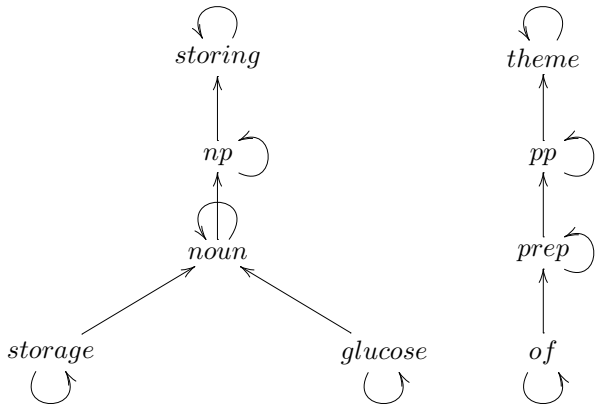


Figure 4.7: Storing Grammar Category Links

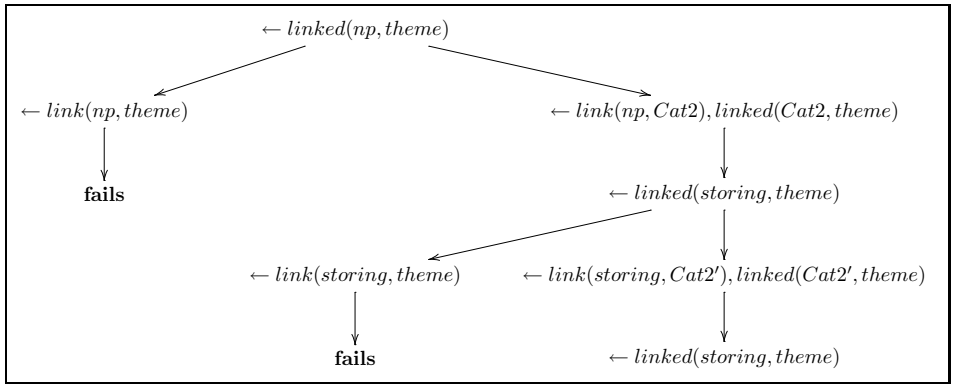


Figure 4.8: Resolution-tree showing the Entrance of an Infinite Loop

predicate is called a sperate tree is created. Basically two trees are created; one for the main goal of $\leftarrow linked(np, theme)$ (figure (1)), and one for the sub-goal of $\leftarrow linked(storing, theme)$ (figure (2)). Since in tabling systems no goal is evaluated more than once, tree (2) sends the answer *NO* - due to the fact that the goal $\leftarrow linked(storing, theme)$ fails - back to its master tree (1). This makes tree (1) to terminate with the answer "NO"- resulting with the derivation-tree shown at the bottom.

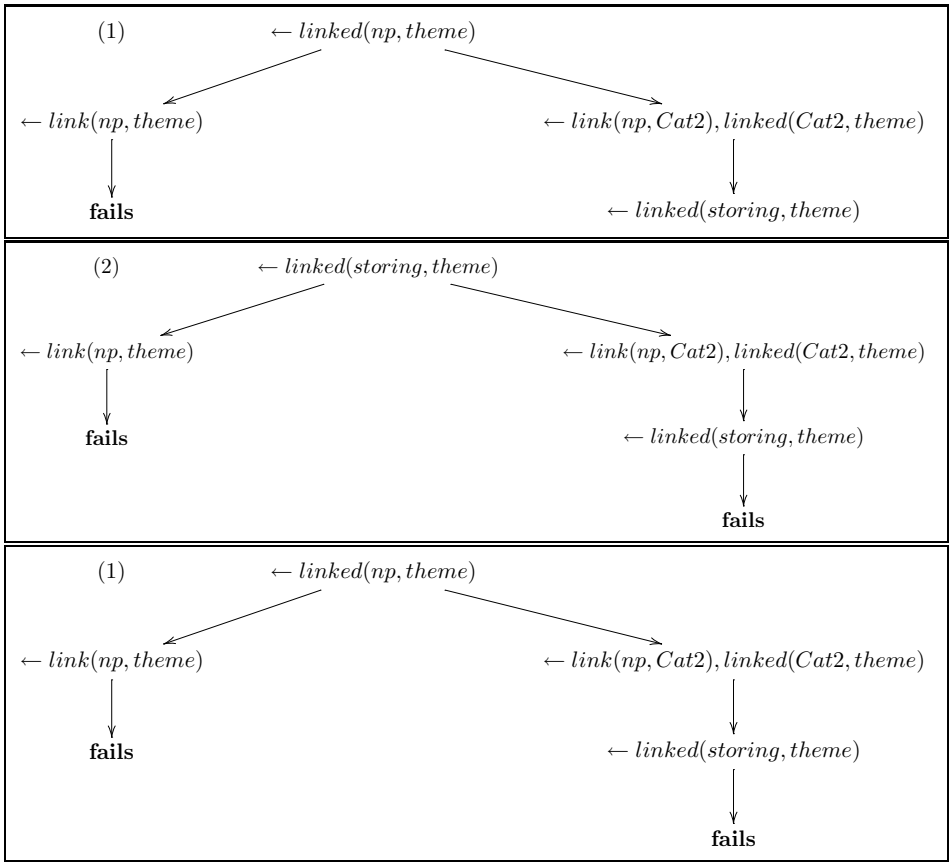


Figure 4.9: Tabled Resolution of the Goal *?−linked(np,theme)*

The recognizer provided by tabled DCG in XSB-Prolog is a variant of Earley's algorithm, i.e. an active chart parsing algorithm.[14] It is easy to see why since each sub-goal with its own tree is an arc, and all of these trees together comprise the chart. The conclusion is that tabled-DCG exhibits all the characteristics of and efficient parser plus it can - as we will see in 4.7.3 - easily be extended with

attributes to evaluate semantics of a well-formed input-string.

One might argue that Earley's parser is still more efficient, since when it finishes we wold have all possible parses of the input on the chart. This is actually also the case with tabled-DCG, since all the goal-servers created will run while the program runs all possible answers can be collect efficiently - since they are already resolved.

4.7.3 Semantics and Attributed Grammars

Extending grammars with attributes, is a method in order to account for (among others) the semantics of syntactic structures. Based on defining a set of attributes for the production rules of a formal grammar; and based on defining rules that determine how these attributes are assigned values; and based on constraints expressed in terms of conditions that must be satisfied by syntactic structures of the language, context-sensitive properties of a language are specified.

Grammars extended with attributes in the way described above, are mainly used for semantic evaluation of syntax structures. The aim of semantic parsing is semantic annotation of sentences or parts of sentences that are recognizable. This basically means that, syntax categories are evaluated with regard to certain attributes. I.e. we think of attribute functions that map syntax categories into attribute-values that are semantic descriptors of the linguistic structures. Grammars extended with attributes, conditions (e.g. semantic constrains) and attribute functions that assign the values of attributes - given a category - are called *attributed grammars*.

The attributed grammar in Table 4.3 is an example of augmenting a CFG in order to specify context-sensitive semantics and conditions. As considered an attribute *Sem* - sem for semantic descriptor - with corresponding evaluation rules for each category is given in order to evaluate semantics. Further semantic constraints (conditions) are given in order to verify the semantics of the underlying linguistic structures. The attribute Sem is a compound structure (feature-structure) which is synthesized. I.e. When considering the parse-tree corresponding to well-formed syntactic structures, the values of the attribute Sem are sent up from descendants to the parents, where they will be included in the semantic descriptors of the parents.

An attributed grammar can be implemented as it is, using DCG-notation. In order to illustrate this consider the following DCG-implementation in Listing 4.9, that implements the attributed grammar considered in Table 4.3.

⟨*storing*⟩	::= ⟨*np(storage)*⟩⟨*theme*⟩ **Sem(**⟨*storing*⟩**)**←[type:transport,elements:[**Sem(**⟨*theme*⟩**)**]]
⟨*theme*⟩	::= ⟨*pp(of)*⟩ **Sem(**⟨*theme*⟩**)**←theme:**Sem(**⟨*pp(of)*⟩**)** **Condition: Sem(**⟨*pp(of)*⟩**) is_a *substance***
⟨*pp(X)*⟩	::= ⟨*p(X)*⟩⟨*np*⟩ **Sem(**⟨*pp(X)*⟩**)**←**Sem(**⟨*np*⟩**)**
⟨*np(X)*⟩	::= ⟨*noun(X)*⟩ **Sem(**⟨*np(X)*⟩**)**←**Sem(**⟨*noun(X)*⟩**)**
⟨*np*⟩	::= ⟨*noun*⟩ **Sem(**⟨*np*⟩**)**←**Sem(**⟨*noun*⟩**)**
⟨*noun*⟩	::= glucose **Sem(**⟨*np*⟩**)**←**Sem(**glucose**)**
⟨*noun(storage)*⟩	::= storage
⟨*p(of)*⟩	::= of

Table 4.3: Attributed Frame-based Grammar

```
1  storing([type:transport,SemTheme]) -->
2          np(storage),theme(SemTheme).
3  theme(theme:SemPP) -->
4          pp(SemPP,of),{subsumed(SemPP,substance)}.
5  pp(SemNP) --> prep,np(SemNP).
6  pp(SemNP,X) --> prep(X),np(SemNP).
7  prep(P) -->[P],{prep([P],[])}.
8  np(Noun)-->noun(Noun).
9  np(SemN,X)-->noun(SemN,X).
10 noun(SemN,N) -->[N],{noun(SemN,[N],[])}.
11 noun(storage,[storage|B],B).
12 noun(glucose,[glucose|B],B).
13 prep([of|B],B).
```

Listing 4.9: Attribute Grammar in DCG

Considering line 1 in Listing 4.9; it is shown how the semantics of *storing* is composed of the semantic value (contribution) of the constituent *theme* and an additional feature-value pair. At line 4 an example of a semantic constraint on the type of the semantic-role filler of *theme* is seen. A sentence having the underlying frame of storing as its meaning, should satisfy this condition. For more details on semantic evaluation and verification see 6 where a detailed explanation of the entire demo, which is essentially a attributed DCG is included.

Chapter 5

Frame-Based Semantic Parsing

In this chapter the design issues of the system are discussed. An account of the requirements and how these requirements are to be met is represented in this chapter; and in the next chapter we will look at the implementation itself in details. First a description of the system in terms of general use-cases is included; what is required for the implementation in terms components and design choices that are made follows afterwards.

Section 5.1 gives an overview of the system and its requirements. Section 5.2 introduces the ontologies used in the system, and shows how their vocabulary can be formalized by a grammar. How to derive context-sensitive frames from ontological concepts is covered in Section 5.3. In Section 5.4 our frame-based grammar is considered, and its properties are explained. Section 5.5 covers ontology-driven search in terms of frame-based semantic descriptors; further how the concept of ontology-driven search is realized here, is described in details.

5.1 The System

Some of the important characteristics of the system suggested here, are considered in the following. The purpose is to provide and overview on what the system is in terms of purpose, functionality and abilities. Before starting the discussion here the system can briefly be described, as a frame-based syntactic-semantic DCG, that is extended with semantic attributes in order to provide semantics of well-formed syntactic structures.

The semantic parser here facilitates two types of semantic representations as the result of semantically parsing domain specific text. The semantic interpretations are essentially twofold. Semantics of a piece of text is on one side the ontological concept, that it maps to in accordance with the frame-based analysis of the text; and on the other side the semantic structure of the underlying frame that is represented in terms of frame-based semantic descriptors (that we will call *semantic descriptor* or *semantic definition*). Semantic descriptors are the direct result of semantic parsing, since the grammar is a frame-based grammar analyzing the text in terms of syntactic-semantic combinations (valence patterns). That the semantics descriptors are the direct result of frame-based semantic parsing, is due to the fact that the semantic elements (frame-elements) will be part of the parse-tree.

Semantic descriptors can easily be converted to corresponding ontological terms, since the vocabulary of the biological ontologies considered here are highly regular. However the role played by the ontologies with respect to, first semantic parsing, and second search (or information retrieval), is not limited to a vocabulary's (a collection of terms) language formalized by a grammar. As we will see the hierarchical structure of the ontology, based on *is-a* relations (the taxonomy) is used in order to impose semantic constraints on frame participants (frame-elements) during parsing, i.e. in order to semantically verify and evaluate input string; and taxonomies additionally are used to assist information retrieval when e.g. no direct hit is in place for a query.

The key points to be made about the characteristics of the system suggested here is first of all how the logical (Prolog) representation of the taxonomy is incorporated into the semantic parser, and is used for assisting information retrieval; second how the power-full resolution system of Prolog, enables us to do the twofold semantic interpretation in one logic program, namely a frame-based DCG semantic parser based on frames; and how it enables us with minimal efforts to implement real ontology driven search on descriptors, based on subsumption supported by the taxonomy.

After this overview general use-cases and an overview on the resources con-

tributing to the system are included below.

5.1.1 Use-Cases

Generally the use-case is of the following: the query (provided by the user) is semantically interpreted, as it is with open text sentences; it will provide a semantic descriptor that will be hold against descriptors found in text in order to provide a semantically relevant result in term of a match. Matching the descriptors is done via subsumption operation on descriptors, i.e. more specialized (detailed) text-descriptors are provided to the more general query-descriptors or vice versa. This is basically the essence of ontology driven querying, since the underlying ontology is used to relate information in accordance to their generality or specialization via their place in the hierarchy.

We have a unique way of translating sentences to go-terms but we can also translate go-terms to definitions (and actually linguistic structures i.e. text), so another use-case is thought based on nodes in the ontology (actually the taxonomy). In this case the ontology's graph is used for navigation through possible result. The user can navigate through the ontology and go up or down in the hierarchy. As the user goes up in the ontology, the terms are more general and depending on the texts available, there will in principle be more results; as the user goes down and the terms gets more specific and narrow, there are less result.[1] This (as we will see in Section 5.5) goes back to subsumption operation on descriptors, since as the query descriptor gets more specific there are less descriptors that can be subsumed by them.

5.1.2 Resources Contributing to the System

There are different resources that contribute to the system. Figure 5.1 gives a rough overview of the involved resources. We view the resources in to main groups, namely ontologies and lexical resources. The contribution of the ontologies is both direct and indirect; direct in form of a taxonomy represented in Prolog that is part of the program; and indirect in terms of how frames included in the system are identified based on the upper-level terms of the ontologies. FrameNet frames contribute indirectly in terms of frames that are customized and extended with ontological semantic type, i.e. ontologies contribute to bio-process frames identified for the system.

[1] This method is best illustrated by GoPubMed where search results are sorted by GO-terms and represented hierarchically corresponding the GO-ontologies.

A more detailed description of these resources and how they are contributing to (are part of) the system is included in the following sections.

5.2 Underlying Ontology

The purpose of this system is deep semantic interpretation. Deep semantic interpretation is meant here as in contrast to shallow semantic interpretation, which is semantic labeling (semantic markup) of text based on linguistic semantics only. Shallow semantics can be either thematic-role labeling based on general linguistic semantics or semantic-role labeling based on frame semantics (e.g. pure FrameNet frames). Deep semantic interpretation is when linguistic semantics is extended with semantic constraints based on formal ontologies. This is the case here as semantic-roles and frames are explicitly constrained by means of formal ontological classes or types. This means that an entity participating in an event with a specific role must be of a certain type satisfying its semantic constraint. The taxonomies are essentially type hierarchies used in this way to verify semantics of linguistic structures.

The semantics here is based formal ontologies from OBO Foundary. The on-

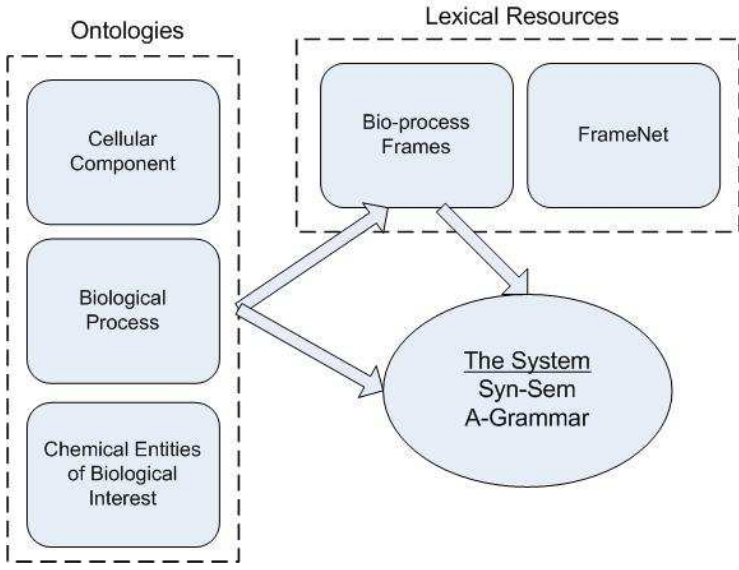


Figure 5.1: Resources and Their Contribution

tologies used are as following:

Biological Process: is part of GO; represents an account of biological processes; is part of OBO Foundry and makes only use of two relations *is_a* and *part_of*

Chemical Entities of Biological Interest: abbreviated to CHEBI, represents chemical entities like insulin and glucose etc.

Cellular Component: is part of GO; represent an account on cells and their components; is part of OBO Foundry and makes only use of two relations *is_a* and *part_of*

It is proper to inform that the ontologies are not used in full scale. They are presented in miniature versions that still adhere to the hierarchies and constraints of the ontologies in full scale.[2] This means that we will not represent any contradiction to the go-ontologies, as they are represented in miniature in our small proof of concept demo.

The relation of primary interest here is *is-a* relation, i.e. we are primarily interested in the taxonomies, that make up the backbone of the ontologies. Only the taxonomies are included in the system, i.e. trees based on is-a relations among ontological term. An example of such miniature taxonomy is depicted in Figure 5.2. Though the taxonomies are not represented in full scale, the idea of using them as axiomatized formal structures in order to verify semantics should be clear.

The underlying terminology (vocabulary), established by the OBO ontologies is highly regular and can be expressed by a grammar. This is an interesting characteristic that we will capitalize on, first in relation to taxonomy-based navigation search, and second in relation to how ontological terms are determined based on frames, i.e. how we can define rules for evaluating term-attributes of frame-based grammar rules. In the following we will look at such a grammar and study some properties of this that will help us in our design.

5.2.1 Ontological Grammar

In this section the emphasis is put on the regularity of the vocabulary (ontological terminology) OBO ontologies, but first some properties of these are considered.

[2]The main reason for representing the ontologies in this way is memory space.

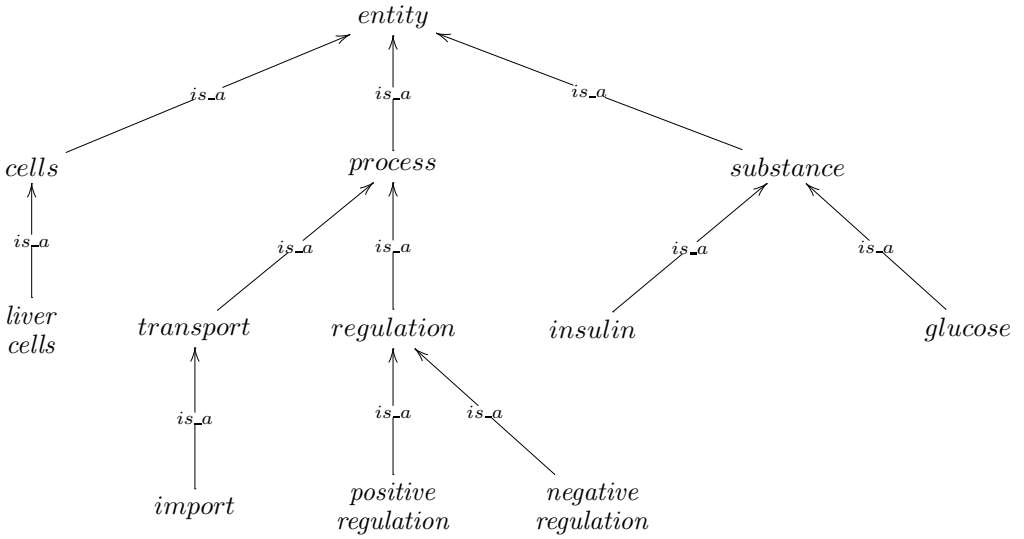


Figure 5.2: GO Taxonomy

OBO ontologies are controlled vocabularies that formalize the medical and biological knowledge. Basically they consist of terms and a finite collection of relations that structure the terms into a hierarchy. OBO.GO (Gene Ontology) which underlies the system here consists of three controlled vocabularies (corresponding to cellular-component, molecular-function and biological-process ontologies) that comprise a collection of terms hierarchally related by means of *is_a* and *part_of* relations.[8] The syntactic structure of the compound terms are regular and can therefore be analyzed (parsed) and interpreted by computers via a CFG. Compound terms' definitions (e.g. frame descriptors same as our sematic descriptors) can automatically be built by attribution of formal grammars on these.[9] Such an grammar is given in Table 5.1. As we see such a grammar merely stipulate the syntactic structures of terms.

The novelty of recognizing that the vocabulary can be modeled with a forman grammar, lies in realizing that the compound terms are composed of atomic terms that correspond to the upper levels of the taxonomies. Most of the compound terms are combinations of *atomic* upper-level terms. By attribution of the rules corresponding to the atomic terms we can have a attributed grammar, that maps ontological terms to corresponding semantic descriptors.

Another further interesting point about the grammar above is that based on syntactical structure of a term the relations among terms can be determined. E.g. *positiv regulation* is subsumed by (is-a) *regulation*, or *glucose import* is a

$\langle term \rangle ::= \langle np \rangle$
$\langle np \rangle ::= \langle np \rangle\ \langle pp \rangle$
$\langle np \rangle ::= \langle noun \rangle$
$\langle pp \rangle ::= \langle prep \rangle \langle np \rangle$
$\langle np \rangle ::= \langle adj \rangle \langle np \rangle$
$\langle noun \rangle ::=$Regulation
$\langle noun \rangle ::=$Transport
$\langle noun \rangle ::=$Import
$\langle noun \rangle ::=$Glucose
$\langle adj \rangle ::=$Positiv
$\langle prep \rangle ::=$of

Table 5.1: Simple GO-vocabulary Grammar

subsumed by *import*. As follows below, when frame definitions with semantic constraints, are defined/identified for ground (or atomic) terms like *regulation transport* etc. any valid specializations of these semantic structures will represent compound subsumed terms of the ground terms.

The key point to have in mind from here, is that a system for semantic parsing based on a taxonomy containing of ground/atomic (upper level) terms only, is in principle as rich with ontological information as a system based on a full-scale taxonomy. Further the system can be based on linguistic properties in terms of frame, identified for the upper-level terms only. The demo implemented in combination with this project is a proof os this fact, where frames that model linguistic appearances of the terms *Regulation* and *Transport* are the only identified frames (with the exception of lacking-frame see 5.3.4 for explanation).

5.3 Ontology-Driven Frames

A sufficiently deep semantic interpretation is a one not based on thematic-roles labeling, but on a more nuanced semantic-role labeling that account for the semantics of natural language expressions, based on the predictions of head words. This is particularly important when text is semantically interpreted based on domain ontologies, because mappings from linguistic expressions to ontological concepts must be established that require more accuracy. Such deep semantic interpretations are provided by semantic frames and collections of pertaining lexical-units, linked with ontological concepts. As indicated before, semantic parsing at this level can be realized by linking central (atomic) concepts with sets of frames representing their semantics in text based on pertaining lexical-units.

In the following the atomic terms of interest are identified, and then corresponding FrameNet frames are selected that can model syntax and semantics of the identified atomic terms in text.

5.3.1 Ontology-Concepts To Frames

Frames are identified for the upper-level atomic terms in the biological-process (Bio-Process) ontology, in order to parse sentences (or phrases) representing the meaning corresponding to these atomic terms, or any terms subsumed by them. Parsing sentences that represent compound-terms, via frames pertaining to the upper-level terms (more general terms), i.e. sub-terms of the upper-level terms results in frame-annotations(see section 3.2 page 16) that can be mapped to the relevant sub-term based on semantic attribution rules introduced in the grammar.

In the following what frames are identified, and why those particular frames are selected is described. But before going on with the process of identifying the frames, the atomic concepts must be identified first. The starting point is essentially the ontological concepts, that are regarded as the atomic concepts. In order to do this, consider the Figure 5.3 that shows a sub-graph of the Bio-Process ontology. In particular the expansion of the ontology's graph down to the level of the term *"Positive Regulation of Glucose Import"* is included here.[3]

The main concepts we will concentrate on here are the concepts of *Biological Regulation* and *Transport*. As we see in Figure 5.4 these to concepts are part of the taxonomical backbone of the upper-level of the ontology.[4] Further we can see in Figure 5.3 that all the concepts of interest for us, are subsumed by these to general concepts.

Frames that capture the meaning of these real life events are identified and rules based on their valence-patterns are introduced into the grammar. The frames of FrameNet are based on corpus evidence and their meaning can be broader or narrower then the intended meaning by the scientific conceptualization of Biological processes. Therefore a selection of FrameNet frames are identified that represent the intended meanings of these biological processes, as sub-concepts of the broader cognitive concepts they represent in open text, (i.e. everything in the targeted natural language). The identified frames are customized (special-

[3]The term *"Positive Regulation of Glucose Import"* is the result of querying the GO Bio-Process ontology with the keywords *insulin glucose*. The search was conducted via AmiGO that is a internet-based search application for Gene ontology database.

[4]We will abstract from *Cellular Process* in this work.

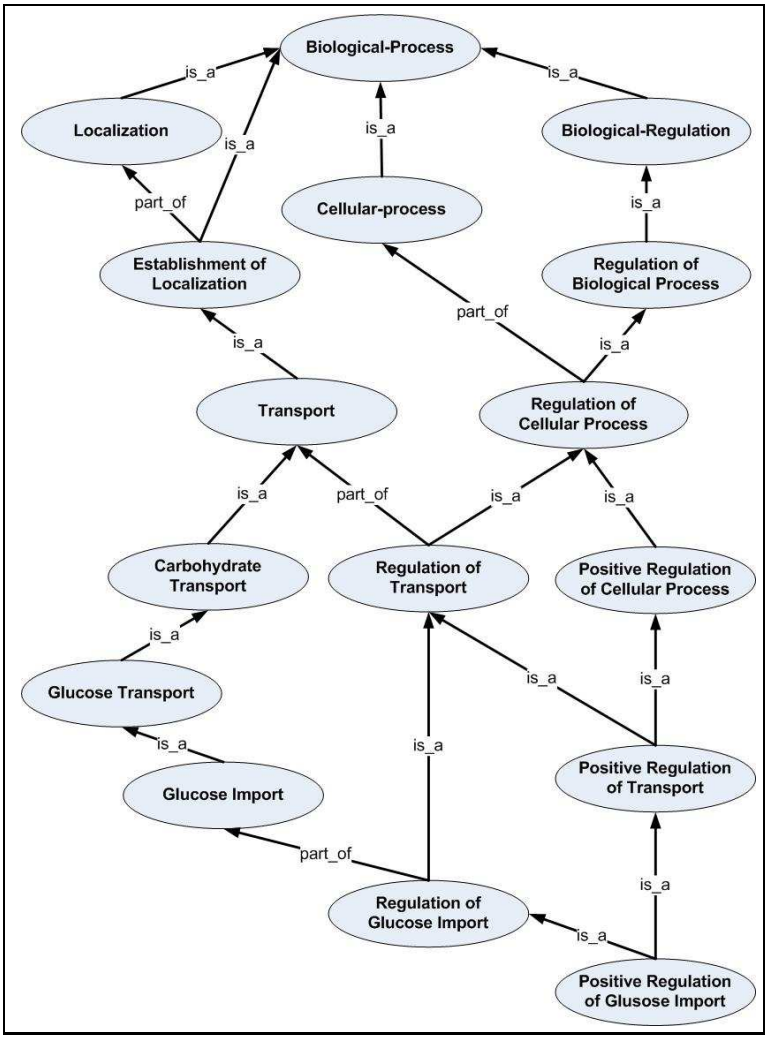


Figure 5.3: GO-Term "Positive Regulation of Glucose Import" placed in the hierarchy

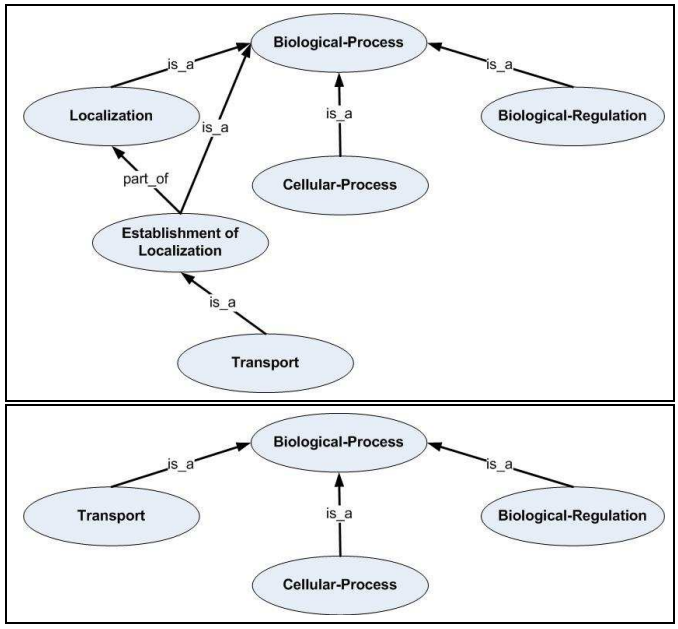


Figure 5.4: GO.Bio-process Upper-level Terms and Corresponding Atomic Terms

ized) with these scientifically described (conceptualized) events in mind. The fusion that results from this process provide us with linguistic level information (Lexical Units and valence-patterns) from one side and formalized structured knowledge bases from the other side and mapping between these two sides. The whole idea with using frames is to be able to map form (text) to semantics (term or nodes in the taxonomy).

In the following we will be identifying frames for each of the to atomic concept identified here, i.e. Regulation and Transport.

5.3.2 Regulation

Two frames that relate to regulation are identified. *Causation* frame describes the event when a *Cause* causes an *Effect*, or when a participant an *Actor* (we will call it Agent) that is involved in the Cause causes an Effect. The idea is that some event or object is responsible for the occurrence of some other event, or establishment of some state.[4] That causation-frame represents the meaning of regulation, may seem improvised and indirect. But if we consider the fact that regulation is also meant, as when a substance is responsible for a certain process it will not be odd to use causation frame. The outlook on the event here (with causation) is the Effect as an event, not the change with respect to a position on a scale.

The other frame identified here that represents the meaning of regulation more directly in the sense of the word, is *Cause-Change-Position-On-Scale*. This frame describes the event that an *Agent* or a *Cause* have an effect on the position of an *Item* on some scale (the *Attribute*). The change can be represented by two values, i.e. a change from an initial value *Value-1* to an end value *Value-2*. It can also be represented with respect to a magnitude of the change (*Difference*) or a direction (*Path*).[4]

The linguistic expressions of these frames are very important, since the attached LUs (Lexical Units) and the pertaining pattern-valences are the main building blocks in our grammar. Some of the most prominent LUs pertaining to causation are *cause.v*, *force.v*, *bring-about.v*; and similarly some of the pertaining LUs for change-position-on-scale are *reduction.n*,*increase.n*, *increase.v*and *decrease.v*.[5]

[5]the letters after dot, indicate part-of-speech, e.g. .n in *reduction.n* for *noun*.

5.3.3 Transport

The picture with the concept transport is more complex. Here two main frames are identified, *Cause-Motion* and *Storing*. Further related frames of Cause-Motion are used to enrich the grammar with variations of this frame. Particularly the sub-frame *Placing* is of interest here and *Bringing* that uses cause-motion. In the demo we will however only make use of Storing and Bringing. The variations introduced by these variants of Cause-motion are both at language level and semantic level since they have different out-looks on the event.

Cause-motion describes the event when an *Agent* causes a *Theme* to move from a Source directly via *Path* to a certain *Goal*. Placing puts emphasis on the Goal of the motion, because LUs of Placing such as *inject.v*, *insert.v*, *insertion.n*, *place.v* and *placement.n*, has focus on the Agent that has control of the Theme all the way to the Goal location. On the other side some cause-motion LUs only focus on the motion caused from the Source without further control (*cast.v*, *throw.v* etc.). With respect to other LUs (e.g. *drag.v*, *push.v*, *force.v*, etc.) the Agent has control of the Theme throughout the motion; and implicit in the meaning of these words is the resistant the Theme in motion must overcome due to some force (e.g. friction) or some obstacle (e.g. cell-membran). Placing differs from Bringing since LUs of the latter (*bear.v*, *bring.v*, *carry.v*, *take.v* etc.) describe the movements where an Agent controls the Path of itself and the Carrier (the agent itself can be the carrier) that bears the Theme. Basically Bringing differs with respect to focus on Carrier that is used to move the Theme.

When we use Storing and Bringing here (in the demo) Theme is the item that is transported; and the Goal is the location where Theme ends up at the end of the transport

5.3.4 Non-ontological Frames

We have to add additional linguistic level frames to account for linguistic assertion that can not directly be mapped to the ontological concepts. In order to illustrate this consider the example of *"lack of insulin increases protein degradation"*. We want to be able to semantically analyze and represent the event where *"lack of"* or ‘*'deficiencies of"* something is participating in an event in any ways, like e.g. Cause of a change-position-on-scale event in this example.

For this purpose we extend our frame-base with the frame *Lacking*, which describes the event of lacking a *Lacked*.[6] This frame contributes to semantic-

[6]Lacking is not a FrameNet frame. A frame from FrameNet that could do the job done

descriptors as well, since it is semantically significant.

5.4 Frame-based Syn-Sem grammar

Through out this paper we have considered frame-based grammatical rules, used as examples etc. But the layers of syntax and semantics that are present in such syn-sem grammar rules, have not been analyzed until now.

Basically the grammar can be viewed in two layers one syntax layer and one semantics layer. While both are based on syntactical structures represented by valence-patterns, the lowest level corresponds to pure syntax of these while the upper level correspond to the semantic tagging of these structures.

Except the layers in the grammar, the role of the ontologies in semantic evaluation of the recognized syntactic structure are studied. How the ontology is used to dismiss or accept the semantic-type of the semantic-role fillers is studied, i.e. sematic verification is studied as well.

5.4.1 Frame-based Grammatical Rules

Following Listing 5.1 will be the basis of the analysis here. The listing shows the two basic components of the system. The attributed frame-based grammar, and a taxonomy (for a graph on the taxonomy see Figure 5.2). We start the analysis by first considering the syntax-layer and the semantic-layer afterwards.

5.4.1.1 Syntax Layer

Syntax layer corresponds to the normal English syntactical categories like $\langle pp \rangle$ and $\langle np \rangle$ etc. The syntax-layer categories are constituents of higher-level syn-sem categories. The grammar rules here are based on valence patterns of LUs, i.e. based on the syntax, they represent the acceptable structures that represent the linguistic realization of the corresponding frame and frame-elements. E.g. a valence pattern for the verb force paired with causation frame, is at the syntax level as $\langle causation \rangle \rightarrow \langle np \rangle, \langle v(force) \rangle, \langle np \rangle$. This essentially means

by our Lacking frame and more is the frame Possession that described the event where “an Owner has (or lacks) a Possession”.[2]

```
process(TTerm)-->transport(TTerm).
process(RTerm)-->regulation(RTerm).
transport(STTerm)-->storing(STTerm).
regulation(CSTerm)-->causation(CSTerm).

start(PTerm)-->process(PTerm).
storing([THTerm,importt])-->[storage],[of],theme(THTerm),[in],goal.
storing([THTerm,importt])-->[storage],[of],theme(THTerm).
theme(N)-->n(N),{subsumed(N,substance)}.
goal-->n(N),{subsumed(N,cell_comp)}.
causation(CSTerm)-->
        agent,[forces],effect(EFTerm),{app([regulation,of],
            EFTerm,CSTerm),!}.
agent-->n(N),{subsumed(N,substance)}.
effect(PTerm)-->process(PTerm).
n(insulin)-->[insulin].
n(glucose)-->[glucose].
n(liver_cells)-->[liver_cells].

class(entity,none).
class(process,entity).
class(transport,process).
class(regulation,process).
class(substance,entity).
class(insulin,substance).
class(glucose,substance).
class(cell_comp,entity).
class(liver_cells,cell_comp).

subsumed(A,B):- class(A,B).
subsumed(A,B):- class(A,C),subsumed(C,B).
subsumed(A,A).
```

Listing 5.1: Tiny Frame–based Grammar

that a sentence of this structure is syntactically an acceptable representant of causation-event in English.

5.4.1.2 Semantic Layer

When the semantic layer that stipulates the acceptable syntactic structure in term of categories that represent the frame-elements (FEs) the rules for e.g. causation will be converted to the following set.

$$\langle causation \rangle \rightarrow \langle agent \rangle \langle v(force) \rangle \langle effect \rangle \tag{5.1}$$
$$\langle agent \rangle \rightarrow \langle np \rangle \tag{5.2}$$
$$\langle effect \rangle \rightarrow \langle storing \rangle \tag{5.3}$$

5.1 stipulates one of the syntactic-semantic realization of the causation-frame in term of syn-sem categories that represent the frame-elements. In return frame-elements are realized by their syntactical constituents like $\langle np \rangle$ etc. If frame-element categories and frame categories are attributed (like in Listing 5.1) we can evaluate them to semantic definitions and ontology-terms as we do (see 6).

5.5 Semantic Descriptors

In the previous sections descriptors were mentioned, in terms of frame semantics and how they were linked to formal ontologies in order to provide semantics. In the following we will take a look at how these are represented in Prolog; and what are the benefits of this representation and the corresponding operations with regard to ontology driven information retrieval.

5.5.1 Descriptors as Feature Structure

Sematic descriptors are represented as feature structures (FS). E.g part of the result of semantically parsing *"insulin forces storage of glucose in lever cells"*, is the FS-based descriptor below.[7]

[7]The other part is the GO-term also generated by the attributed grammar.

$$(a)\begin{bmatrix}\text{regulation} & \\ \begin{bmatrix}\text{Agent} & \text{insulin} \\ \text{Effect} & \begin{bmatrix}\text{transport} & \\ \begin{bmatrix}\text{Theme} & \text{glucose} \\ \text{Goal} & \text{lever-cells}\end{bmatrix} & \end{bmatrix}\end{bmatrix} & \end{bmatrix}$$

In the following we will benefit from what FS essentially are, and from some operations applicable to FS. First consider the examples below that illustrates what real ontology driven information retrieval is; and what this means in terms of FS-based semantic descriptors of ours here.

Given the result from the example above (descriptor (a)) that could be queried with the following FS-descriptor below (descriptor (b)), which could represent the semantics of e.g. *"insulin action"* or *"insulin responsive action"*. Since the information contained in the query (b) is contained in the text-descriptor (a) the system must respond with the matching descriptor (a). In this case FS (a) is subsumed (b) since (a) is a specialization of (b), which is more general. This is of course as we will see a trivial form of specialization since (a) contains merely more information (more feature-value pairs).

$$(b)\begin{bmatrix}\text{regulation} & \\ \begin{bmatrix}\text{Agent} & \text{insulin}\end{bmatrix} & \end{bmatrix}$$

We need a real subsumption operation based on the underlaying ontological subsumption relations. Recall that a class A in a taxonomy subsumes a class B, iff $B\,is\_a\,A$ or if $A\,subsumes\,C$ and $B\,is\_a\,C$. Basically we want to extend the the trivial subsumption from above with a *typed* subsumption, that incorporates the type hierarchy of the underlying taxonomy. Based on this, querying the system with the FS (c) below will yield the result (a) again because *transport* is subsumed by *process*. Descriptor (c) could be the descriptor of the query *"insulin responsive processes"* or *"biological process caused by insulin"*.

$$(c)\begin{bmatrix}\text{regulation} & \\ \begin{bmatrix}\text{Agent} & \text{insulin} \\ \text{Effect} & \\ \begin{bmatrix}\text{process}\end{bmatrix} & \end{bmatrix} & \end{bmatrix}$$

The idea is as illustrated above to be able to go from a general query to a specialized results that correspond to the hierarchical relation of values. By representing our descriptors as FS and implementing FS-subsumption extended whit taxonomies (type hierarchies) we have fulfilled the promise of ontology driven information retrieval.

5.5.1.1 FS Implementation

Feature structures can essentially be represented as attribute value matrices. We choose a matrix representation in the following that convert the descriptor (1) given above to as follows.

$$\begin{bmatrix} \text{regulation} & \\ \begin{bmatrix} \text{Agent} & \text{insulin} \\ \text{Effect} & \begin{bmatrix} \text{transport} & \\ \begin{bmatrix} \text{Theme} & \text{glucose} \\ \text{Goal} & \text{lever-cells} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{Type} & \text{regulation} \\ \text{Elements} & \begin{bmatrix} \text{Agent} & \text{insulin} \\ \text{Effect} & \begin{bmatrix} \text{Type} & \text{transport} \\ \text{Elements} & \begin{bmatrix} \text{Theme} & \text{glucose} \\ \text{Goal} & \text{lever-cells} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

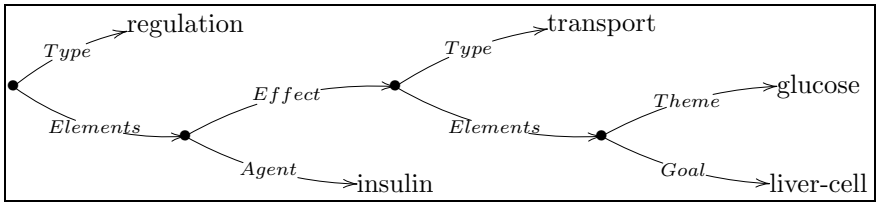
This of course do not essentially change any thing (it hardly needs to be mentioned). The former form is preferred by linguist and is more concise, but the latter form helps in terms of the implementation.[8] There are to partial operations on FS, namely subsumption and unification, based on the ideas of combing two FSs and comparing two FSs. [9] In the following subsumption will introduce, and unification left out, as unification is introduced already (**??** on page **??**) and will not be using it here.

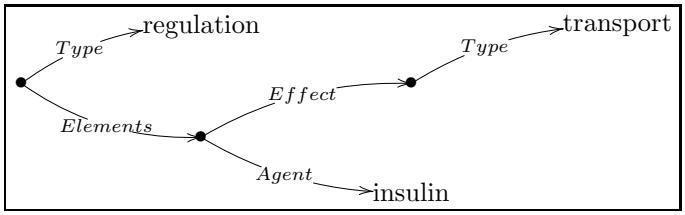
First consider the intuitive definition of ordinary (not typed FS) FS subsumption. Feature structures are sets of properties.Sets of properties are compared in order to see if they convey compatible information. A feature-structure F1 subsumes ($\sqsubseteq$) another feature-structure F2, iff all the information that is contained

[8] We have adapted an even simpler (more matrix shaped) representation in the demo.
[9] This are partial operations since they are not defined for all pairs of FS-descriptors.

in F1 is also contained in F2. This correspond to the first (trivial) example shown above. Before a formal definition of subsumption for the *typed feature structures* of ours here is considered, consider what FS really are and how this helps to understand typed subsumption of FS.

FS are basically multidimensional matrices of feature-value pairs that can be represented as lists of attribute-value pairs. FS can also be viewed as Directed Acyclic Graphs (DAG), which they represent. The features correspond to labeled directed edges and the nodes the labeled edges lead, to are the values of the features. Consider the following graph representations of FS-descriptors (a) and (c) from above. We will say that the first graph is subsumed by the second graph, and it is clear that the patterns out of the roots are the same; and further that nodes in the matching subgraphs are either the same, or the nodes from the second graph are generalizations (super-classes) of the corresponding nodes in the first graph.





A DAG is a recursive data structures and a recursive definition of subsumption can easily be elaborated into the implementation of subsumption operation itself. If each path in $dag1$ going out from the root leading to a node $n1$ leads to a $n2$ in $dag2$ that $n1$ subsumes (i.e. $n1 \sqsubseteq n2$) then $dag1 \sqsubseteq dag2$. So to paths are the same if they contain the same sequence of edges, and as far subsumption goes the nodes along a path in the subsumed dag must be either same as the corresponding node in the subsuming dag, or be subsumed by the nodes of the subsuming dag.

We have a *typed feature structures* since ground values are concepts from the domain of the used ontologies and compound values are typed FS themselves. In order to introduce the notion of typed FS consider the following definition of a *type hierarchy*.

Definition 5.1 (Type Hierarchy) [15] A type hierarchy is a finite bounded complete partial order $\langle \text{Type}, \sqsubseteq \rangle$.

A taxonomy as (e.g. the one depicted in Figure 5.2) is a type hierarchy since the finite set of classes, with the partial order *is_a* and class *entity* that is the supremum (any path in the tree lead to *entity*), is a complete partial order. Hence a taxonomy combined with FS in the way suggested in Definition 5.2 provide us with a typed FS.

Definition 5.2 (typed feature structure) [15] A typed feature structure is defined on a finite set of features **Feat** and a type hierarchy $\langle \mathbf{Type}, \sqsubseteq \rangle$. It is a tuple $\langle Q, r, \delta, \theta \rangle$ where:

- Q is a finite set of nodes,
- $r \in Q$ r is the root node
- $\theta : Q \rightarrow \mathbf{Type}$ is a partial typing function
- $\delta : Q \times \mathbf{Feat} \rightarrow Q$ is a partial feature value function

A feature structure F' subsumes F (written $F' \sqsubseteq F$) if each path from the root π in F' leads to the same node in F as in F', or leads to q' in F' and q in F and $q\, is\_a\, q'$. This is the definition of subsumption corresponding to ground nodes (leafs), if the nodes led to are FS themselves the node fs' in F' and fs in F; fs' should subsume fs ($fs' \sqsubseteq fs$). A general mathematical definition of subsumption for DAG and non-DAG FS is given in [15] as follows:

Definition 5.3 (Subsumption) [15] F' subsumes F, written $F' \sqsubseteq F$, if and only if:

- $\pi \equiv F\pi'$ implies $\pi \equiv F'\pi'$
- $P_F(\pi) = t$ implies PF0 $P_{F'}(\pi) = t'$ and $t' \sqsubseteq t$

$\pi \equiv F\pi'$ notates path equivalence for F with respect to the paths π and π' (i.e. the two paths lead to the same node from the root $\delta(r, \pi) = \delta(r, \pi')$). $P_F(\pi) = \sigma$ means that the type on the path π in F is σ, i.e. $\theta(\delta(r, \pi)) = \sigma$.[15]

Consider the following implementation of subsumption in accordance to the definition considered above. We have a plain feature-value pair matrix representation of our feature structures. With this simple representation we are able

```
1  subsumes(Dag,Dag):-!.
2  subsumes([],_):-!.
3  subsumes(Value1,Value2):-
4        class(Value1,_),
5        class(Value2,_),!,
6        subsumed(Value2,Value1).
7  subsumes([Feature:Value|Dag1],Dag2):-
8        subsume_aux(Feature,Value,Dag2,DagReast),
9        subsumes(Dag1,DagReast).

11 subsume_aux(Feature,Value1,[Feature:Value2|DagReast],
       DagReast):- !,
12       subsumes(Value1,Value2).
13 subsume_aux(Feature,Value,[FeatValue|Descendants],[FeatValue
       |DagReast]):-!,
14       subsume_aux(Feature,Value,Descendants,DagReast).
```

Listing 5.2: Feature Structure Subsumption

to implement a quite simple and efficient implementation of feature structure subsumption.

If $F' \sqsubseteq F$, each path in F' must subsume the corresponding path in F; the clause at line 7 in Listing 5.2 basically defines this. *subsume_aux* predicate defined by clauses at lines 11 13 check whether a path out from the root of F' initiated by the a particular arrow labeled *Feature* subsumes any corresponding path in F; this is done by a depth-first tree-walk (tree-traversal). As indicated above a ground value (left-nodes) q' subsumes a ground value q ($q' \sqsubseteq q$) iff the qis_aq' this is expressed by the *subsumes* clause at line 3 where qis_aq' is tested by *subsumed(Value2,Value1)*(at line 3). [10] Further note that the implementation is for DAG restricted feature structures only.

An additional useful feature that should be provided by the implementation, is to be able to determine whether a DAG subsumes any subgraph of another DAG. This *subgraph subsumption* facility is useful when we look for a concept in a descriptor that is an element of a bigger concept. E.g when we look for a descriptor of a *transport* concept that might be an element of a *regulation* concept. The fact that we look for subsumed subgraph implies that the system must provide the descriptor of the concept *import*,which is and element of a *regulation* concept, if we look for the subsuming concept *transport*.

[10] For efficiency reasons we constrain the the test *subsumed(Value2,Value1)* to ontology concepts only(via lines 4 and 5).

```
1 subsumes_subdag(Dag1,Dag2):-subsumes(Dag1,Dag2),!.
2 subsumes_subdag(Dag1,[_:Value|_]):- subsumes_subdag(Dag1,
      Value),!.
3 subsumes_subdag(Dag1,[_|Dag2]):- subsumes_subdag(Dag1,Dag2)
      ,!.
```

Listing 5.3: Feature Structure Unification

$$
(a)\begin{bmatrix}\text{regulation} & \\ \begin{bmatrix}\text{Agent} & \text{insulin} \\ \text{Effect} & \begin{bmatrix}\text{import} & \\ \begin{bmatrix}\text{Theme} & \text{glucose} \\ \text{Goal} & \text{lever-cells}\end{bmatrix} & \end{bmatrix}\end{bmatrix} & \end{bmatrix}
$$

$$
(b)\begin{bmatrix}\text{transport} & \\ \begin{bmatrix}\text{Theme} & \text{substance}\end{bmatrix} & \end{bmatrix}
$$

$$
(c)\begin{bmatrix}\text{import} & \\ \begin{bmatrix}\text{Theme} & \text{glucose} \\ \text{Goal} & \text{lever-cells}\end{bmatrix} & \end{bmatrix}
$$

The system facilitating subgraph-subsumption queried with descriptor (b) above given descriptor (a) should provide us with the answer (c).

In Listing 5.3 included three definite clauses of *subsumes−subdag* that define *subgraph subsumption* operation on our DAG feature structures. In the program a very simple depth-first search is made to determine whether the firs argument *Dag1* subsumes any subgraph of second argument *Dag2*.[11]

[11] This program could easily be extended all subsumed subgraphs.

Chapter 6

Implementation

In this chapter issues concerning the implementation of a small demo is considered. First an overview on the semantic representation, which is an account on how the semantic descriptors and ontological terms are represented, is provided. Afterwards the DCG-based implementation is considered in details and its functionality is demonstrated with few selected tests. This is divided in sections corresponding to the main ontological terms and the corresponding grammar in terms of the pertaining frames, and other utilities accompanying the program. At the end a general discussion of the implementation will follow that focuses on the strength and weaknesses of the implementation.

In Section 6.1 how semantic representations, which are the result of parsing is explained. In Section 6.2 and Section 6.3 frames pertaining to the ontological concepts of regulation and transport are considered. Necessary linguistic grammar rules and frames included in the system are covered in Section 6.4. In section 6.5 the go-grammar that maps ontological terms to descriptors is described and demonstrated. Section 6.6 includes some tests on descriptor operations, that was extensively explained in last chapter. Finally in the last section a brief discussion of the weaknesses and strengths of the system is covered.

```
1 ?- start(Desc, Term, [lack, of, insulin, forces, increase, of,
      protein, degradation], []).

3 Desc = [type:regulation, cause:[type:lack, lacked:insulin],
      effect:[type:regulation, difference:positive, item:[protein
      , degradation]]]
4 Term = [insulin,[regulation,[protein, degradation]]];

6 no
```

Listing 6.1: Lack of insulin forces increase of protein degradation.

6.1 Semantic Representations

As mentioned in previous chapter, we conduct two simultaneous semantic evaluations of text; one in terms of the corresponding ontological term (GO-term in particular), and one in terms of frame-based descriptors. In order to illustrate this consider the following semantic evaluation of the sentence *"lack of insulin forces increase of protein degradation"* in Listing 6.1. The resulting descriptor from this example is represented below in the more reader friendly FS-notation.

$$\begin{bmatrix} \text{Type} & \text{regulation} & \\ \text{Cause} & \begin{bmatrix} \text{Type} & \text{lack} \\ \text{Lacked} & \text{insulin} \end{bmatrix} & \\ \text{Effect} & \begin{bmatrix} \text{Type} & \text{regulation} \\ \text{Difference} & \text{positive} \\ \text{Item} & \text{protein-degradation} \end{bmatrix} & \end{bmatrix}$$

This basically as expected describes the main event of regulation in terms of its Cause, and Effect; forcing or causing some event/process is interpreted as regulation, based on causation frame. The Cause is a lacking situation where the Lacked item is insulin; the Effect is a regulation process too with positive Difference, indicating increase in the Item that is being regulated.

The term representation is in the form of lists with in lists. In particular *[insulin,[regulation,[protein, degradation]]]* represents an specialization of the term Regulation, namely Regulation of Protein-degradation with insulin as the responsable substance.

In the following the formats of semantic representations (descriptors and terms) corresponding to each frame are considered in detail, but first an overview is provided.

6.1.1 Frame-based Semantic Descriptors

Descriptors have the general format given below.

$$\begin{bmatrix} \text{Type} & \textit{go-term} \\ \text{Semantic-role-1} & \textit{value-1} \\ \ldots & \\ \text{Semantic-role-k} & \textit{value-k} \end{bmatrix}$$

Each frame maps to a go-term which is the value of the feature Type in the descriptor; each Semantic-role denotes a frame-element pertaining to the particular frame underlying the considered interpretation. E.g. in the example above we had the semantic-roles (or relations) Cause and Effect, that are elements of the Causation frame which was evoked by the sentence, i.e. represented the semantics of the sentence. More details on the actual format of descriptors, with regard to what are the semantic-roles pertaining to each frame, will be introduced when each of these frames are considered.

6.1.2 Ontological Terms

The way the go-terms are represented as lists within lists, is an implementation issue that keeps the program simple and ensures generative capabilities of the program. By generative capabilities we mean the ability to go from terms to possible linguistic forms (text) and descriptors. Further information on how these are built with regard to the each grammar rule pertaining to the frames will be considered when the corresponding frame is considered.

6.2 Regulation Frames

In this implementation two semantic frames are associated with biological regulation (GO-term Regulation), namely *change-position-on-scale* and *causation*.

```
1 agent(agent:N,N)-->
2         np(N),{subsumed(N, substance),\+subsumed(N, glucose)}.

4 cause(cause:LDef,LTerm)-->
5         lacking(LDef,LTerm).
6 cause(cause:PCDef,PCTerm)-->
7         position_change_scale1(PCDef,PCTerm).
```

Listing 6.2: Regulation Frame–elements Agent and Cause

When descriptors are built based on the linguistic models these two frames represent, the descriptors will have the following basic abstract structure that is realized with different variations depending on the frames and the pertaining valence-based rules.

$$\begin{bmatrix} \text{Type} & \text{regulation} \\ \text{Cause} & \textit{cause-value} \\ \text{Agent} & \textit{agent-value} \\ \ldots & \end{bmatrix}$$

First let us consider the similarities between the implementation of these two frames in terms of the shared elements Cause and Agent.

The Agent can be any subsumed class of the class *substance* in the ontology, but it can not be subsumed by the class *glucose*. [1] The Cause can be events like *"lack of some-substance"* or *"some-substance deficiency"* (like e.g. insulin deficiency); and *"reduction of some-substance"* or *"increase of some-substance"* can be causes of regulation processes as well. These are respectively defined by the two alternatives of *cuase* at lines 5 and 7 in Listing 6.2.

The semantic contributions of Agent and Cuase in terms of descriptors are feature-value pairs (i.e. *cause:LDef* and *agent:N*). Cause-values are compound values, i.e. FS-descriptors; and agent-values are atomic-values, i.e. ontological terms. I terms of what they contribute to the ontological term evaluation; they contribute with the agent (e.g. insulin) in both cases so terms of e.g. the following form are built *[agent−value,[regulation ,[some−process]].*

[1] Glucose may in reality participate in the cause of some biological regulation process (i.e. be the Agent), but right now it is not allowed for glucose to fill the role of Agent.

6.2.1 Change-Position-on-Scale Frame

In the following change-position-on-scale frame, as is called *position_change_scale* in the implementation, is considered. Change-position-scale frame has two semantic-roles in addition to Cause and Agent (introduced above), namely Difference and Item.

Regulation of biological processes can be changes of positions on scales with different dimensions, e.g. magnitude, rate and frequency etc. On the linguistic level change can be described generally with words like *increase* or *decrease*, that are respectively translated to *positive* and *negative* here. When considering the dimension of the change with respect to a process in GO-ontology, it is implicit in the ontology. The interpretations here (positive and negative) relies on the way GO-terms/classes are defined e.g. like *"positive regulation of glucose import"* and the corresponding *"negative regulation of glucose import"* as sub-terms of *"regulation of glucose import"*. These terms are interpreted as increase of the import-process, and decrease of the import-process respectively.[2] Basically positive and negative are atomic values of the semantic-role Difference. The additional semantic-role pertaining to the frame here is Item, which is the item that changes position on a scale with some abstract/implicit dimension.

In order to have more control on what subsets of linguistic appearances, of this frame can e.g. be the Cause and Effect of in regulation events (introduced above in 6.2), we will distinguish between rules belonging to the lexical-units of *reduction.n* and *increase.n*, and rules belonging to the lexical-units of *decrease.v* and *increase.v*. The first set of rules given in Listing 6.3 pertain to *position_change_scale1* and are based on valences of *reduction.n* and *increase.n*. As we have seen (in Listing 6.2) *position_change_scale1* can be the Cause of regulation processes along with lacking events.

Item (Listing 6.3 at lines 6-7) which is shared by *position_change_scale* (Listing 6.5) and *position_change_scale1* , either can be an ontological term, or it can be an linguistic representation of an transport-process.

Listing 6.4 shows tests of *position_change_scale1* used first (at line 1) for semantic parsing, and second for generation of text based on the term *"protein degradation"*(at line 8).

Since reduction or increase of some substance can be the Cause or Effect of an causation event, or Cause of an position-change event; while the defini-

[2]Positive and negative might in situation be the inverse of what we understand by them namely increase and decrease respectively. But this is considered to be a matter for ontologist to precisely represent what the correct interpretation is.

```
position_change_scale1([difference:negative,type:regulation,
    IDef],ITerm)-->
        [reduction],[of],item(IDef,ITerm).
position_change_scale1([difference:positive,type:regulation,
    IDef],ITerm)-->
        [increase],[of],item(IDef,ITerm).

item(item:NP,NP)-->np(NP).
item(item:PDef,Term)-->transport(PDef,Term).
```

Listing 6.3: Change–Position–on–Scale Frame1

```
?- position_change_scale1(Def,TM,[increase,of,protein,
    degradation],[]).

Def = [difference : positive,type : regulation,item : [
    protein,degradation]]
TM = [protein,degradation];

no

?- position_change_scale1(Def,[protein,degradation],ST,[]).

Def = [difference : positive,type : regulation,item : [
    protein,degradation]]
ST = [increase,of,protein,degradation];

Def = [difference : negative,type : regulation,item : [
    protein,degradation]]
ST = [reduction,of,protein,degradation];

no
```

Listing 6.4: Test of position_change_scale1

```
1  position_change_scale([difference:positive,type:regulation,
       IDef,CDef],
2            [CTerm,[regulation,ITerm]]) -->
3                 cause(CDef,CTerm),[increases],item(IDef,ITerm).
4  position_change_scale([difference:positive,type:regulation,
       IDef,ADef],
5            [ATerm,[regulation,ITerm]])-->
6                 agent(ADef,ATerm),[increases],item(IDef,ITerm).
7  position_change_scale([difference:negative,type:regulation,
       IDef,CDef],
8            [CTerm,[regulation,ITerm]])-->
9                 cause(CDef,CTerm),[decreases],item(IDef,ITerm).
10 position_change_scale([difference:positive,type:regulation,
       IDef,ADef],
11           [ATerm,[regulation,ITerm]])-->
12                agent(ADef,ATerm),[decreases],item(IDef,ITerm).
```

Listing 6.5: Change–Position–on–Scale Frame2

```
1  ?- position_change_scale(Def,Term,[lack,of,insulin,
       increases,protein,degradation],[]).

3  Def = [difference:positive,type:regulation,item:[
       protein,degradation],cause:[type:lack,lacked:
       insulin]]
4  Term = [insulin,[regulation,[protein,degradation]]];

6  no
```

Listing 6.6: position_change_scale Test1

tion(descriptor) describes the event in details, the term returned is merely the Item-term. This is due to (as we saw above in 6.2) the fact Cause and Agent both contribute with single terms, to the term build-up of regulation frames. This is the same for when *position_change_scale1* is the Effect of Causation. An example illustrating the latter is given in Listing 6.10.

The second set of rules given in Listing 6.5 pertain to *position_change_scale* and are based on valences of *reduction.v* and *increase.v* lexical-units.

Listing 6.6 shows a test of *position_change_scale* used for semantic parsing; and Listing 6.7 shows a test for generation of text based on a go-terms. Among the texts generated there are non-valid sentences, e.g. while the sentence *lack of insulin increases protein degradation* is valid, the sentence *reduction of insulin*

```
1  ?- position_change_scale(Def,[insulin,[regulation,[protein,
       degradation]]],ST,[]).

3  Def = [difference : positive,type : regulation,item : [
       protein,degradation],cause : [type : lack,lacked :
       insulin]]
4  Sentence = [lack,of,insulin,increases,protein,degradation];

6  Def = [difference : negative,type : regulation,item : [
       protein,degradation],cause : [difference : negative,type
       : regulation,item : insulin]]
7  Sentence = [reduction,of,insulin,decreases,protein,
       degradation];
```

Listing 6.7: position_change_scale Test2

```
1  | ?- position_change_scale([difference:positive,type:
       regulation,item:[protein,de
2  gradation],cause:[type:lack,lacked:insulin]],TM,ST,[]).

4  Term = [insulin,[regulation,[protein,degradation]]]
5  Sentence = [insulin,deficiency,increases,protein,degradation
       ];

7  Term = [insulin,[regulation,[protein,degradation]]]
8  Sentence = [lack,of,insulin,increases,protein,degradation];

10 yes
```

Listing 6.8: position_change_scale Test3

```
1 causation([type:regulation,ADef,EDef],[ATerm,[regulation,
      EFTerm]])-->
2           agent(ADef,ATerm),[forces],effect(EDef,EFTerm).
3 causation([type:regulation,CDef,EDef],[CTerm,[regulation,
      EFTerm]])-->
4           cause(CDef,CTerm),[forces],effect(EDef,EFTerm).

6 effect(effect:PDef,PTerm)-->transport(PDef,PTerm).
7 effect(effect:PDef,PTerm)-->position_change_scale1(PDef,
      PTerm).
```

Listing 6.9: Causation Frame

decreases protein degradation is not valid and contradictory to the first sentence. The system is not geared to verify the sentences it builds based on go-terms at these level. But mapping (generation) from descriptors to text as is tested in the Listing 6.8 yields valid sentences only.

6.2.2 Causation

When natural language representations of Regulation can be modeled with Causation frame the Effect of the regulation process is in focus as well as Cause or Agent. The Effect can be a transport or position-change event.

Listing 6.10 shows an example of semantically parsing the the sentence *"insulin forces reduction of protein degradation"*. At a closer look on the result we see that even the effect of the main regulation event is a regulation event itself, but the term it translated to (*[insulin,[regulation,[protein,degradation]]]*) denote the single biological regulation event as it should. The nested regulation event in the descriptor reflect the right linguistic perspective this event; while the term is a mapping of the term to a node the ontological hierarchy.

6.3 Transport Frames

Transport descriptors have the following general form. And in certain cases the Carrier and Goal sematic-roles with the corresponding fillers will be added to the descriptors. Two frames pertaining to transport are included Bringing and Storing. Listing 6.11 shows the rules pertaining to these two transport-frames.

```
| ?- causation(Def,TM,[insulin, forces, reduction, of,
    protein, degradation],[]).

Def = [type : regulation,agent : insulin,effect : [
    difference : negative,type :
regulation,item : [protein,degradation]]]
TM = [insulin,[regulation,[protein,degradation]]];

no
```

Listing 6.10: Test causation

$$\begin{bmatrix} \text{Type} & \text{transport} \\ \text{Theme} & \textit{theme-value} \\ \text{Goal} & \textit{goal-value} \\ .... & \end{bmatrix}$$

6.4 Linguistic-level Frames and the Ontology

The frame-based grammar here is assisted by some very simple natural language phrasal rules defining some simple set of noun-phrases, and the linguistic-level frame of lacking.

6.4.1 Lacking Frame

We have already seen the Lacking frame in action, which models linguistic expressions denoting the event of lacking. The underlying meaning of lacking, though very important semantically is not covered by any ontological concept since it pertains to propositional semantics of text. Lacking here is defined on the two lexical-units of *deficiency.n* and *lack.n* as illustrated in Listing 6.12. It has the single element Lacked and will have descriptors consisting of *type : lack* and Lacked having a term as value. It contributes only lacked term to any term build-up.

```
storing([type:transport,TDef,GDef],[THTerm,importt])-->
        [storage],[of],theme(TDef,THTerm),[in],goal(GDef,_).
storing([type:transport,TDef],[THTerm,importt])-->
        [storage],[of],theme(TDef,THTerm).

bringing([type:transport,TDef],[THTerm,transport])-->
        [intake],[of],theme(TDef,THTerm).
bringing([type:transport,CRDef,TDef,GDef],[THTerm,transport
    ])-->
        carrier(CRDef,_),[carry],theme(TDef,THTerm),[in],
            goal(GDef,_).

theme(theme:N,N)-->
        np(N),{subsumed(N,substance)}.
goal(goal:N,N)-->
        n(N),{subsumed(N,cell_comp)}.
carrier(carrier:N,N)-->
        np(N),{subsumed(N,substance),\subsumed(N,glucose)}.
```

Listing 6.11: Transport Frames

```
lacking([type:lack,LDef],LTerm)-->
        [lack],[of],lacked(LDef,LTerm).
lacking([type:lack,LDef],LTerm)-->
        lacked(LDef,LTerm),[deficiency].
lacked(lacked:N,N)-->np(N),{subsumed(N,substance)}.
```

Listing 6.12: Lacking Frames

```
1 n(N)-->[N],{class(N,_)}.
2 np(N)-->n(N).
3 np([N1,N2])-->np(N1),np(N2).
4 adj(positive)-->[positive].
5 adj(negative)-->[negative].
```

Listing 6.13: Phrase–grammar

6.4.2 Phrase Grammar and Ontology

A noun in our grammar is an atomic term from the underlying taxonomy. Compound terms are the subset of noun-phrases that are conjunctions of terms, are called np here. Additionally we have positive and negative adjectives. The very simple phrase-grammar assisting our frames is included in 6.13

6.5 GO Grammar

When search is based on the ontological hierarchy we have to have a means of determining corresponding semantic definitions (descriptor) of each node in the hierarchy. For this purpose a grammar with precedence (see Listing 6.14) is defined that can map terms to descriptors and vice versa.

Listing 6.15 show two test of GO-grammar; firs test where the term *"insulin regulation of glucose import"* is translated to its corresponding semantic descriptor; and second test where a semantic descriptor is translated to the corresponding GO-terms. Note that the answer at line 10 in Listing 6.15 subsumes the first answer at line 8, this is due to the fact that import processes get the type transport that is the super-term, in their semantic definitions (see Listing 6.14 line 21).

6.6 Descriptor Implementation

Descriptor querying facilities are already described in detail (in Section 5.5 on page 73) therefor this part of the implementation will not be considered further. Instead we will look at some tests.

Let us consider the two simple tests included in Listing 6.16 that test the oper-

```
1  :- table go_term/3.
2  :- [ontology].
3  :- [phrase_grammar].
4  go_term(TDef)-->go_term1(TDef).
5  go_term1([difference:A|TDef])-->
6              adj(A),go_term2(TDef).
7  go_term1(TDef)-->go_term2(TDef).
8  go_term1(TDef2)-->go_term4(TDef2).
9  go_term1(TDef2)-->go_term5(TDef2).

11 go_term2([agent:TDef1|TDef2])-->
12             go_term5(TDef1),go_term3(TDef2).
13 go_term2(TDef2)-->go_term3(TDef2).

15 go_term3([type:regulation])-->[regulation].
16 go_term3([type:regulation,effect:TDef])-->
17             [regulation],go_term4(TDef).

19 go_term4([type:transport,theme:TDef])-->
20             go_term5(TDef),[transport].
21 go_term4([type:transport,theme:TDef])-->
22             go_term5(TDef),[importt].
23 go_term4([type:transport])-->[transport].
24 go_term4([type:importt])-->[importt].

26 go_term5(NP)-->n(NP).
```

Listing 6.14: Go–grammar

```
1  ?- go_term(Def,[insulin,regulation,glucose,importt],[]).

3  Def = [agent : insulin,type : regulation,effect : [type :
       transport ,theme : glucose]];

5  no
6  | ?- go_term([difference:positive,agent:insulin,type:
       regulation,effect:[type:transport,theme:glucose]],X,[]).

8  X = [positive,insulin,regulation,glucose,importt];

10 X = [positive,insulin,regulation,glucose,transport];

12 no
```

Listing 6.15: Go–grammar Test

```
1 ?- subsumes([type:regulation, agent:insulin, effect:[type:
     process]],[type:regulation, agent:insulin, effect:[type:
     transport, theme:glucose, goal:liver_cells]]).

3 yes
4 ?- subsumes_subdag([type:transport, theme:substance],[type:
     regulation, agent:insulin, effect:[type:transport, theme:
     glucose, goal:liver_cells]]).

6 yes
```

Listing 6.16: Descriptor Operations Test

ations, we defined on descriptors in Section 5.5. The first test (at line 1) that tests the subsumption operation, is the same as querying the system with e.g. ‘*“insulin regulation”*. The second test (at line 4) that tests subsumes-subdag (explained in 5.5), corresponds to the case where we are looking for a *“transport of any substance”* event, that is part of a compound event like e.g. *“insulin forces storage of glucose”*.

6.7 Strength and Weaknesses

The strength of the system is first and foremost the accuracy of the semantic interpretations it produces. Its weakness is that, it is case-based and not more general in its nature.

One issue illustrated by examples above (in 6.2.1), is when text is generated based on a term. Some of these generated text pieces (as we saw) are invalid assertions in the considered domain. But on the other hand the GO-grammar (see Section 6.5) translates go-terms to semantics descriptors quit accurately; and in return these can be used to generate valid text, that can e.g. be used to search the web with using traditional key-word based search. Searching the web with a set of semantically related key-words must in principle result in semantically relevant results, since traditional key-word search is based on distances of appearances of the key-words in text. .

Chapter 7

Conclusion

In this chapter a brief comparison of the method suggested here and an alternative method, for semantic parsing is included. Further what the next step would be, if a full-scale semantic parser based on frames were to be realized, is considered. At the end the project is summarized, in terms of achievements and lessons of the project, for a session of ending remarks.

7.1 Discussion

In this section we would have a brief discussion on an alternative method compared with the method suggested in this document - and used in this project. We take a look at weaknesses and advantages of frame-based semantic parsing compared with the alternative, that is thematic-role labeling via generative grammars.

Further we will see how linguistics based approaches like semantic frames - that is the right and hard way of approaching semantics - can be combined with generic methods that are more desirable for computer systems.

7.1.1 Frame-Based Grammar

One of the weaknesses of frame based grammar is that, it is case based. I.e. it is based on syntactic-semantic properties of single words. This causes reputations in terms of rules for syntactical structures that would be avoided if we had a general CFG. But this will not be an issue if the syntactic-semantic properties of words are represented in lexicons with well designed, well specified format, and the grammar is a Head-driven Phrase Structure Grammar (HPSG)[10]. HPSG represent a generic model on natural languages, since it stipulates general phrase-structures in terms of e.g. FS-descriptor defining linguistic properties of the constituents.

The advantage of our frame-based approach to semantics, is essentially its accuracy, which comes with the price of being costly to implement.

7.1.2 Generative Grammar

When general thematic-roles are used for analysis of semantics based on syntax only, the probability for erroneous semantic interpretation is big. This is first and foremost due to ambiguity of grammars, and deficiencies inherited in the method of assigning roles based on syntax only. Syntactic and semantic properties of words must be considered in any consistent method for analysis of natural language semantics. Further there is another issue with regard to thematic-role labeling that concerns the nature of thematic-roles themselves. Thematic roles are not context sensitive and in principle cannot easily be extended with meaningful semantic constraints.

Though generative approaches based only on syntax, are not able to give a sound analysis of semantics, they are more likely to be implemented since less efforts are required for their implementations compared with our frame-based approach. But frame-based approach is still preferred over the alternative since with even limited numbers of frames and lexical-units, lots of semantically domain relevant information can be processed and generated.

7.2 The way forward

We will look at different approaches each rooted in their own fields of work here, that seem to be in contrast with each other but can be combined in order to realize feasible NLP systems.

7.2.1 Linguistics Approach vs Generative Approach

A linguistics based, case-sensitive lexicographical approach to semantics of natural languages is a must. By linguistics based approach, we mean methods concerned with formalizations of natural language properties (syntax and semantics) that are realistic and context-sensitive in their view on language.

While in linguistics the objective is a comprehensive formalization, the comprehensiveness is not necessarily required for computer systems. Depending on the applications, the systems may require more or less comprehensive formalizations. In computer science the trend is to look for generalizations that formalize the domains of interest - here natural language syntax and semantics - in a concise way. Without generative formalizations implementations of e.g. semantic processors are not feasible. By generative we mean that not everything is explicitly stated, but the underlying formalizations give room for generation of required information. Examples of generative formalizations are e.g. ontological grammars as well as ontologies themselves.

One might think that the the linguistics based approach is the opposite of the computer-science algorithmic approach. But as the latest efforts in the computational linguistics field shows, these two approaches are not necessarily in conflict. With the emergence and evolvement of lexical resources like WordNet, FrameNet etc. and generic linguistic formalizations (like the idea of HPSG), it is possible to have the best of both worlds. It is crucial to drag on both approaches in order to build the next generation of natural language processors, that can cope with the demands of the information-age.

7.2.2 The Way Forward

The role played by domain ontologies as the basis for semantics - basis for semantic analysis, representation and search - is the single most important issue for systems like the one suggested here. Ontologies must be incorporated in the grammars in oder to be able to impose semantic constraints and facilitate real ontology-driven search. In order to fulfill this requirement DCG-notation extended with semantic constraints in combination with logical representation of the underlying ontologies were chosen. But a generic modular architecture based on a concise formalizations is clearly preferred over the status quo.

FS-based HPSG in combination with well documented, reliable lexical resources that give a comprehensive account of syntactic-semantic properties of words, is the way forward. We have already considered how FS-based grammar rules

work e.g. with regard to valence etc. FS-notation were in fact used in order to make the case for the use of semantic frames. The XML-based architecture of FrameNet essentially represents its syntactic-semantic data in a format similar to FS. But a generic architecture is not easily achieved without resolving some design and implementation issues. First and foremost, frames from FrameNet must be extended with domain specific semantic constraints, and semantic verification based on these constrains should be facilitated based on ontologies. The project essentially consists of specification and elaboration of the right format for lexical data that represent valence patterns along with semantic constrains, and building systems on this format which do the semantic parsing an verification.

Collecting and specifying the frames is likely to take much effort and time. But that job can be done by ontologist and linguists. Efforts on specifying biomedical specific frames are in motion as I write now. With regard to this the interested reader can read about the project BioFrameNet, which is "domain-specific FrameNet extension".[17]

7.3 Final Remarks

The project documented here by this report has shown that semantic frame-based analysis of natural languages, can provide a mapping from text to semantics, that is reliable and significant. The method derives domain specific semantic structures (frames) from ontological terms, that enables us to analyze, determine and represent domain specific semantics of text. The extend of semantic depth provided with the suggested method is beyond what can be expected from a generative natural language grammar based on thematic-role labeling, even extended with controlled vocabularies. The semantic interpretations achieved by the suggested method are supported by semantic verifications based on formal consistent domain specific taxonomies. Mappings from text to semantics in terms of ontological terms and semantic descriptors are supported by well documented, realistic linguistic patterns with evidence in large collection of text. The semantic interpretation provided goes beyond mappings from text to ontological terms only, they even provide propositional semantics that can be used for reasoning.

The Prolog-based technique suggested for building the semantic parser facilitates several modes of use. The system can be used for semantic parsing and correspondingly ontology driven information retrieval based on queries with text. It also allows for querying the system with ontological terms that based on a grammar are mapped into semantic descriptors. This feature of the system al-

lows for navigational style, hierarchy based search of information, based on the hierarchy of the ontology.

The amounts of work required for a realistic implementing based on the method suggested here is the main obstinate. Though I have shown that even with limited amount of syntactic-semantic lexical entries extended with ontological types a vast amount of semantically significant pieces of information can be gathered (parsed). But the implementation method should be replaced with a more generic implementation, where the parsing and semantic verification is separated from the grammar, which is structured in a formally specified format.

The main result here is the research that is documented by this paper. The research has brought about a through analysis of semantics in natural language based on the idea of frame semantics. Further how semantic frames are to be extended and used, for ontology-driven deep semantic interpretation of domain specific content is shown. Issues concerning implementation of formal ontology-based semantic parsers, with frame-based grammars are analyzed and resolved in this project. In particular methods for defining or identifying frames based on formal domain ontologies are illustrated. Further how these frames are extended with ontological type constraints are explained. A demo proving the concept of frame-based domain specific semantic interpretation is implemented. This demo shows how ontology driven information retrieval can be facilitated.

Appendix A

Appendix A

In this you will find the code of the Prolog-program I have work with. These are divided in to Sections A.1 includes the implementations of the parsers work on in association with Chapter 4. These are files with .pl extension, i.e. they are SWI-Prolog files.

Sections A.2 includes the code for the demo created in this project. This files are with .P extension, i.e. they are XSB-Prolog files.

A.1 Parsers

A.1.1 leftcorner_recognizer.pl

Listing A.1: leftcorner_recognizer.pl

```
1 :- op(700,xfx,--->).
2 leftcorner_recognizer(StartSymbol,Wordlist):-
3         scan(StartSymbol,Wordlist,[]).

5 scan(Prediction,[Word|Wordlist],RmWordlist):-
6         lex(Word,LexCat),
```

```
7           complete(Prediction,LexCat,Wordlist,RmWordlist).

9   complete(Prediction,Prediction,Wordlist,Wordlist).
10  complete(Prediction,CompleteCat,Wordlist,RmWordlist) :-
11          LHS ---> [CompleteCat|Predictions],%the notation with
                operator ---> is same as rule(LHS,[SubCat|
                Predictions]).
12          predict(Predictions,Wordlist,RmWordlist1),
13          complete(Prediction,LHS,RmWordlist1,RmWordlist).

15  predict([],Wordlist,Wordlist).
16  predict([Prediction|Predictions],Wordlist,RmWordlist):-
17          scan(Prediction,Wordlist,RmWordlist1),
18          predict(Predictions,RmWordlist1,RmWordlist).

20  :- [frameGrammar].
```

Listing A.1: leftcorner_recognizer.pl

A.1.2 bottom_up_acp.pl

Listing A.2: bottom_up_acp.pl

```
1   :- op(700, xfx, --->).
2   :-dynamic arc/4.
3   :-dynamic word/3.
4   :-[frameGrammar].

6   bottom_up_active_chart(Start_Symbol,Input_String) :-
7           clean_chart,
8           init_chart(Input_String, 0),
9           init_agenda(Agenda),
10          process_agenda(Agenda),
11          length(Input_String, N),
12          arc(N,0 , [],Start_Symbol).

14  predict(arc(E,S,[],Cat1),Agenda,Agenda1):-
15          findall(arc(E,S,RHS,Cat2),Cat2--->[Cat1|RHS],NewArcs),
16          append(NewArcs,Agenda,Agenda1).
17  predict(_,Agenda,Agenda).

19  complete(arc(E1,S1,[],Cat1),Agenda,Agenda1):-
20          findall(arc(E1,S2,RHS,Cat2),arc(S1,S2,[Cat1|RHS],Cat2),
                NewArcs),
21          append(NewArcs,Agenda,Agenda1).
22  complete(_,Agenda,Agenda).
```

```
24 scan(arc(E1,S1,[Cat2|RHS],Cat1),Agenda,Agenda1):-
25       findall(arc(E2,S1,RHS,Cat1),arc(E2,E1,[],Cat2),NewArcs),
26       append(NewArcs,Agenda,Agenda1).
27 scan(_,Agenda,Agenda).

29 init_agenda(Agenda) :-
30       findall(arc(E, S, [], Cat),
31       (word(Word,S,E),lex(Word,Cat)),Agenda).

33 init_chart([], _).
34 init_chart([Word|String], E) :-
35       NewE is E + 1,
36       assert(word(Word,E,NewE)),
37       init_chart(String, NewE).

39 process_agenda([]).
40 process_agenda([Arc|Agenda]) :-
41       add_to_chart(Arc,Agenda,NewAgenda),
42       process_agenda(NewAgenda).

44 add_to_chart(Arc,Agenda1,Agenda4):-
45       \+Arc,!,
46       assertz(Arc),
47       predict(Arc,Agenda1,Agenda2),
48       complete(Arc,Agenda2,Agenda3),
49       scan(Arc,Agenda3,Agenda4).
50 add_to_chart(_,Agenda,Agenda).


53 clean_chart:-
54       retractall(arc(_,_,_,_)),
55       retractall(word(_,_,_)).
```

Listing A.2: bottom_up_acp.pl

A.1.3 Grammars

Listing A.3: frameGrammar.pl

```
2 storing ---> [np,theme].
3 theme ---> [pp].
4 np ---> [noun].
5 pp ---> [prep,np].
6 lex(storage,noun).
```

```
lex(glucose,noun).
lex(of,prep).

% link(np,storing).
% link(pp,theme).
% link(noun,np).
% link(noun,storing).
% link(prep,pp).
% link(prep,theme).
% link(X,X).
```

Listing A.3: frameGrammar.pl

Listing A.4: expGrammar.pl

```
%% Author:
%% Date: 31-08-2007

e ---> [t].
e ---> [e,pl,t].
t ---> [p].
t ---> [t,ml,p].
lex(a,p).
lex(+,pl).
lex(*,ml).
```

Listing A.4: expGrammar.pl

A.2 Demo

A.2.1 frame_grammar.P

Listing A.5: frame_grammar.P

```

:-table process/4.
:-table regulation/4.

:-[ontology].
:-[utility].
:-[phrase_grammar].
:-[position_change_scale].
:-[causation].
:-[storing].
```

```
:-[bringing].
:-[lacking].

start(Def,PTerm)-->process(Def,PTerm).
process(Def,TTerm)-->transport(Def,TTerm).
process(Def,RTerm)-->regulation(Def,RTerm).
process(Def,RTerm)-->lacking(Def,RTerm).
transport(Def,STTerm)-->storing(Def,STTerm).
transport(Def,STTerm)-->bringing(Def,STTerm).
regulation(Def,CSTerm)-->causation(Def,CSTerm).
regulation(Def,PSTerm)-->position_change_scale(Def,PSTerm).
regulation(Def,PSTerm)-->position_change_scale1(Def,PSTerm).
```

Listing A.5: frame_grammar.P

A.2.2 ontology.P

Listing A.6: ontology.P

```
:-table subsumed/2.

class(entity,none).
class(process,entity).
class(transport,process).
class(regulation,process).
class(degradation,process).
class(substance,entity).
class(hormone,substance).
class(insulin,hormone).
class(glucose,substance).
class(cell_comp,entity).
class(liver_cells,cell_comp).
class(fat_cells,cell_comp).
class(protein,substance).

subsumed(A,B):- class(A,B).
subsumed(A,B):- class(A,C),subsumed(C,B).
subsumed(A,A).
```

Listing A.6: ontology.P

A.2.3 phrase_grammar.P

Listing A.7: phrase_grammar.P

```
:- table np/3.

n(N)-->[N],{class(N,_)}.
np(N)-->n(N).
np([N1,N2])-->np(N1),np(N2).
adj(positive)-->[positive].
adj(negative)-->[negative].
```

Listing A.7: phrase_grammar.P

A.2.4 lacking.P

Listing A.8: lacking.P

```
:-table lacking/4.

lacking([type:lack,LDef],LTerm)-->[lack],[of],lacked(LDef,
    LTerm).
lacking([type:lack,LDef],LTerm)-->lacked(LDef,LTerm),[
    deficiency].
lacked(lacked:N,N)-->np(N),{subsumed(N,substance)}.
```

Listing A.8: lacking.P

A.2.5 bringing.P

Listing A.9: bringing.P

```

bringing([type:transport,TDef],[THTerm,transport])-->
                    [intake],[of],theme(TDef,THTerm).
bringing([type:transport,CRDef,TDef,GDef],[THTerm,transport])
    -->
                    carrier(CRDef,_),[carry],theme(TDef,THTerm),[
                        in],goal(GDef,_).

carrier(carrier:N,N)-->np(N),{subsumed(N,substance),\subsumed(
    N,glucose)}.
```

Listing A.9: bringing.P

A.2.6 storing.P

Listing A.10: storing.P

```
storing([type:transport,TDef,GDef],[THTerm,importt])-->
            [storage],[of],theme(TDef,THTerm),[in],goal(GDef,_).
storing([type:transport,TDef],[THTerm,importt])-->
            [storage],[of],theme(TDef,THTerm).

theme(theme:N,N)-->n(N),{subsumed(N,substance)}.
goal(goal:N,N)-->n(N),{subsumed(N,cell_comp)}.
```

Listing A.10: storing.P

A.2.7 position_change_scale.P

Listing A.11: position_change_scale.P

```
:-table position_change_scale/4.
:-table position_change_scale1/4.

position_change_scale1([difference:negative,type:regulation,
    IDef],ITerm)-->
                            [reduction],[of],item(IDef,ITerm).
position_change_scale1([difference:positive,type:regulation,
    IDef],ITerm)-->
                            [increase],[of],item(IDef,ITerm).

position_change_scale([difference:positive,type:regulation,
    IDef,CDef],[CTerm,[regulation,ITerm]])-->
                            cause(CDef,CTerm),[increases],item(
                                IDef,ITerm).
position_change_scale([difference:positive,type:regulation,
    IDef,ADef],[ATerm,[regulation,ITerm]])-->
                            agent(ADef,ATerm),[increases],item(
                                IDef,ITerm).
position_change_scale([difference:negative,type:regulation,
    IDef,CDef],[CTerm,[regulation,ITerm]])-->
                            cause(CDef,CTerm),[decreases],item(
                                IDef,ITerm).
position_change_scale([difference:positive,type:regulation,
    IDef,ADef],[ATerm,[regulation,ITerm]])-->
                            agent(ADef,ATerm),[decreases],item(
                                IDef,ITerm).

item(item:NP,NP)-->np(NP).
```

```
19 item(item:PDef,Term)-->transport(PDef,Term).
```

Listing A.11: position_change_scale.P

A.2.8 causation.P

Listing A.12: causation.P

```
2  :-table causation/4.
3  :-table cause/4.

5  causation([type:regulation,ADef,EDef],[ATerm,[regulation,
       EFTerm]])-->
6                  agent(ADef,ATerm),[forces],effect(EDef,EFTerm).
7  causation([type:regulation,CDef,EDef],[CTerm,[regulation,
       EFTerm]])-->
8                  cause(CDef,CTerm),[forces],effect(EDef,EFTerm).

10 agent(agent:N,N)-->np(N),{subsumed(N,substance),\+subsumed(N,
       glucose)}.
11 cause(cause:LDef,LTerm)-->lacking(LDef,LTerm).
12 cause(cause:PCDef,PCTerm)-->position_change_scale1(PCDef,
       PCTerm).

14 effect(effect:PDef,PTerm)-->transport(PDef,PTerm).
15 effect(effect:PDef,PTerm)-->position_change_scale1(PDef,PTerm)
       .
```

Listing A.12: causation.P

A.2.9 go_grammar.P

Listing A.13: go_grammar.P

```
1  :-table go_term/3.
2  :-[ontology].
3  :-[phrase_grammar].
4  go_term(TDef)-->go_term1(TDef).
5  go_term1([difference:A|TDef])-->
6              adj(A),go_term2(TDef).
7  go_term1(TDef)-->go_term2(TDef).
8  go_term1(TDef2)-->go_term4(TDef2).
9  go_term1(TDef2)-->go_term5(TDef2).
```

```
11 go_term2([agent:TDef1|TDef2])-->
12           go_term5(TDef1),go_term3(TDef2).
13 go_term2(TDef2)-->go_term3(TDef2).

15 go_term3([type:regulation])-->[regulation].
16 go_term3([type:regulation,effect:TDef])-->
17           [regulation],go_term4(TDef).

19 go_term4([type:transport,theme:TDef])-->
20           go_term5(TDef),[transport].
21 go_term4([type:importt,theme:TDef])-->
22           go_term5(TDef),[importt].
23 go_term4([type:transport])-->[transport].
24 go_term4([type:importt])-->[importt].

26 go_term5(NP)-->n(NP).
```

Listing A.13: go_grammar.P

A.2.10 subsumption.P

Listing A.14: subsumption.P

```
1  :- table subsumed/2.
2  :- table subsumes_subdag/2.
3  :-[ontology].

5  subsumes(Dag,Dag):-!.
6  subsumes([],_):-!.
7  subsumes(Value1,Value2):-
8            class(Value1,_),
9            class(Value2,_),!,
10           subsumed(Value2,Value1).
11 subsumes([Feature:Value|Dag1],Dag2):-
12           subsume_aux(Feature,Value,Dag2,DagReast),
13           subsumes(Dag1,DagReast).

15 subsume_aux(Feature,Value1,[Feature:Value2|DagReast],DagReast)
       :- !,
16           subsumes(Value1,Value2).
17 subsume_aux(Feature,Value,[FeatValue|Descendants],[FeatValue|
       DagReast]):- !,
18           subsume_aux(Feature,Value,Descendants,DagReast).

20 subsumes_subdag(Dag1,Dag2):-subsumes(Dag1,Dag2),!.
```

```
21 subsumes_subdag(Dag1,[_:Value|_]):- subsumes_subdag(Dag1,Value
       ),!.
22 subsumes_subdag(Dag1,[_|Dag2]):- subsumes_subdag(Dag1,Dag2).
```

Listing A.14: subsumption.P

Bibliography

[1] Fillmore C.J., Johnson C.R., Petruck M.R.L. (2003). Background to Framenet. International Journal of Lexicography, Vol 16.3: 235-250. *DTV Article Database Service.* Oxford University Press. (25 June 2007).

[2] Baker C.F., Fillmore C.J., Cronin B. (2003). The Structure of the Framenet Database. International Journal of Lexicography, vol. 16.3: 281-296. *DTV Article Database Service.* Oxford University Press. (25 June 2007).

[3] Ruppenhofer J., Ellsworth M., Petruck M.R., Johnson C.R. (2005). FrameNet: Theory and Practice. ICSI Berkeley. http://framenet.icsi.berkeley.edu. (15 March 2007)

[4] FrameNet Project ICSI Berkeley. (2007). http://framenet.icsi.berkeley.edu. (15 July 2007)

[5] Gene Ontology. (2007). http://www.geneontology.org/ (15 July 2007)

[6] Wikipedia, the free encyclopedia. (2007). Insulin. http://en.wikipedia.org/wiki/Insulin. (15 April 2007)

[7] Temperley D., Sleator D., Lafferty J. (2007). Link Grammar. http://www.link.cs.cmu.edu/link/index.html. (15 April 2007)

[8] Smith B., Ceuster W., Klagges B., Köhler J., Kumar A., Lomax J., Mungall C., Neuhaus F., Rector A.L., Rosse C.(2005). Relations in biomedical ontologies. Genome Biology 2005. http://bmc.ub.uni-potsdam.de/gb-2005-6-5-r46/gb-2005-6-5-r46.pdf. (20 July 2007)

[9] Mungall C.J. (2004). Obol: integrating language and meaning in bio-ontologies. Comparative and Functional Genomics Comparative and Functional Genomics. vol. 5, no. 6-7, pp. 509-520. http://www.fruitfly.org/∼cjm/obol/doc/Mungall_CFG_2004.pdf. (20 July 2007)

[10] Sag I.A., Wasow T. (1999). Syntatic Theory A Formal Introduction. CSLI Lecture Notes. http://www.cs.um.edu.mt/ mros/ftp/download/sw99.pdf (15 June 2007)

[11] Hopcroft J.E., Motwani R., Ullman J.D. (2003). Intoduction to Automata Theory, Languages, and Computation. Pearson Addison Wesley. ISBN: 0321210298.

[12] Blackburn P., Striegnitz K. (2002). Natural Language Processing Techniques in Prolog. http://www.coli.uni-saarland.de/ kris/nlp-with-prolog/html/ (15 June 2007)

[13] Voss M.(2004). Improving Upon Earley's Parsing Algorithm in Prolog. http://www.ai.uga.edu/mc/ProNTo/Voss.pdf (15 June 2007)

[14] Warren D.S.(1999). Programming in Tabled Prolog. http://www.cs.sunysb.edu/ warren/xsbbook/book.html (15 July 2007)

[15] Copestake A. (2001). Appendix: definitions of typed feature structures. Natural Language Engineering. citeseer.ist.psu.edu/copestake01appendix.html (15 August 2007)

[16] Guarino N., Welty C.A., Staab S., Studer R.(2004) An Overview of OntoClean. Handbook on Ontologies. International Handbooks on Information Systems. Springer. ISBN 3-540-40834-7, 151-172

[17] Dolbey A., Ellsworth M., Scheffczyk J.(2006) BioFrameNet:A Domain-specific FrameNet Extension with Links to Biomedical Ontologies. KR-MED 2006: Biomedical Ontology in Action. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-222/krmed2006-p10.pdf (15 May 2007)