Domain Knowledge for Natural Language Processing

Andrei Mikheev

HCRC Language Technology Group
University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW, Scotland, UK
E-mail: Andrei.Mikheev@ed.ac.uk

January 17, 1996

Abstract

In this paper we describe the organization and contents of the knowledge base (KB) developed for the processing of patient discharge summaries (PDSs)—letters sent by a hospital consultant to a patient's own doctor. The processing system itself can be seen as a specialization of the Generic Information Extraction System outlined by Hobbs (Hobbs 1993). As it is stated in (Jacobs *et al* 1993) among the most succesfull information extraction systems "the main differentiator is not in the basic processing algorithms but in the way that knowledge ends up getting assigned to various system components". The KB is the major component in the processing of natural language by our system and includes both the conceptual and lexical knowledge represented in a uniform way. Conceptual knowledge in the KB is built and organized in a top-down fashion and explicitly targeted to represent information which was identified by domain experts to be important. Linguistic knowledge is built around the conceptual structures in a bottom-up and empirically driven way and encodes the actual textual representations of particular classes of conceptual structures. The fundamental idea behind this approach is grouping of lexical knowledge around conceptual structures by means of lexico-semantic constructions, which cover entire classes of NL expressions and are pre-packed with rules for deterministic interpretation of these expressions into conceptual schemata of the domain. Although this KB is explicitly targeted to the medical domain and its sublanguage we argue that the approach advocated in this paper can be applied to multi-domain NLP. On the basis of some example texts and the system output, we will illustrate how acquired and represented background knowledge is used in processing to yield the desired interpretations.

keywords: knowledge-based natural language processing; medical informatics; information extraction; sublanguage semantic parsing; knowledge representation.

1 The Task

For the last few years we have been involved in the development of a system for the processing of patient discharge summaries (PDSs)—letters sent by a hospital consultant to a patient's own doctor, after that patient has spent some time in the hospital. The PDS corpus available at our site consists of some five hundred discharge letters which are all in the area of heart disorders. Each letter includes a standard header with structured administrative information as patient's name, date of birth, address, main diagnosis, etc. and a body text. A typical example of a PDS letter is represented in figure 1.

UNIT NUMBER : JX9990
PATIENT NAME : MCDOOGAL
SEX : M
DOB : 12.07.1924
ADMITTED :18.10.1994
DISCHARGED :18.10.1994
MAIN DIAGNOSIS : G34z.... Other chronic IHD NOS

Dear Doctor Georgian

This 70 year old man was admitted from the Happy Valley Hospital on 28th January 1991 for angina pectoris. There was a seven year history of myocardial ischaemiac disease with a diaphragmatic infarction in December 1990 followed by unstable angina requiring re-admission to hospital.

Selective coronary arteriography and left ventriculography were performed on 29.1.91. This revealed a severe left myocardial infarct and a high grade stenosis at the origin of the circumflex coronary artery. The stenosis appeared to be involving the left main coronary artery. The right coronary artery was totally occluded but filled well via collaterals.

Mr.McDoogal was transferred back to the Happy Valley Hospital on 30th January 1991 and should his symptoms continue Mr McDoogal would be willing to consider surgical revascularisation.

Figure 1: A Patient Discharge Summary (PDS).

The need for the processing of the PDSs emerged from the fact that textual information stored in the computer cannot be used for the purposes of automatic summarization and question answering. So, for example, to find out how many patients were admitted for angina pectoris last month, a separate hand-entered data-base record should be created for each discharge summary. In some cases "flat" data structures supported by the record-field format of data bases will not be able to express the relations between data items in different fields (Onyshkevych 93). Thus, for example, in order to correlate patient symptoms and performed investigations in response to that symptoms cross-references between the fields should be supported. The PDSs are in a well-defined area, that of coronary disorders.

But the level of understanding needed is fairly deep: the system is intended to capture information that may be of interest to consultants, general practitioners, registrars, nurses, researchers, etc. For example:

- **administrative data**: patient's name, sex, age; date and mode of admission and discharge; etc;
- **pre-hospitalization medical data**: risk factors, family and personal antecedents;
- **hospitalization info**: reason for admission; clinical information; diagnostic procedures (type, date and results); therapeutic procedures (type, date and location);
- **post-hospitalization prescriptions**: discharge medications; planned diagnostic and therapeutic procedures;

All this information is represented in free text form in the discharge letters and the task of the system is to identify a certain piece of information in the text and represent it in a form, which can be accessible by other programs, for question-answering, summarization and some other purposes. Such representation should be built not from textual but rather from logical entities, which are arranged into structures with formally interpretable semantics. A long standing tradition in medical informatics (see, for example, Campbell 1992) is to use the Conceptual Graph (CG) notation (Sowa 1984) to represent the logical contents of NL texts. This notation is argued to be more perspicuous for domain experts than, for example, predicate calculus and well tailored to represent many phenomena of NL. Not making claims on the expressiveness of the Conceptual Graph theory we use the CG notation as a fancy way to represent the facts extracted from PDSs.

In brief, a conceptual graph consists of concepts linked to each other with conceptual relations. A concept node is represented in square brackets: [CONCEPT]. A relation node is represented in round brackets: →(relation)→. A concept node includes two fields – a type field and a referential field and they are separated by a semicolon. For example, [PERSON:x]— represents a concept of type person. By default it is assigned a singular distributed denotation which corresponds in language to the article "a". The variable "x" in the referential field fixes the distributed denotation, and it is now possible to make references to "a particular person x" by [PERSON: *x]. Another important feature for us is the representation of the object/attribute/value triple: [OBJECT]→(attr)→[COLOR]→(val)→[RED] and the attribute/value pair can be contacted into [COLOR: @red]. Let us consider, for example, a sentence:

The right coronary artery was totally occluded.

This sentence can be logically represented in the CG notation as follows:

```
[med-condition-prop: x26]
    →(char)→[med-condition: x25]
                    →(kind)→[occlusion]
                    →(attr)→[extent:@total]
                    →(loc)→[bodypart: #rca]
    →(attr)→[status:@fact]
```

This introduces the concept MED-CONDITION-PROP which represents the proposition that a medical condition of type "occlusion" was *loc*ated in the bodypart known as "right coronary artery". This medical condition is linked to the proposition node as its structural component by means of the relation of characterization (char). The proposition also has an attribute of its status which represents whether the whole proposition is a fact, advice, guess or a possible event in the future. Concepts are assigned with their identifiers by which they can be referred to: [med-condition: x25]. Some concepts, as, for example, body-parts, are defined as unique entities since every person has only one right coronary artery or myocardium. This is represented by special definite references: [bodypart: #rca]. The user interface of the system allows for verification and validation of the extracted from the text facts. Figure 2 shows the working screen after the PDS analysis. Displayed in the right top window CG representation of the facts can be edited if the captured information is not right.

Although at the discourse level the PDS corpus does not present too many problems other than from coreference resolution, the sentence level processing is not an easy one. The problem that arises is the following. From the linguistic point of view the corpus presents a number of challenges. Standard problems of lexical and structural ambiguity, compound nominals, metonymy are accompanied by challenges of real-world language usage as misspellings, grammatically incorrect structures and contracted sentences. Apart from considerable natural language expertise, this task also requires the system to be endowed with a lot of specialized domain knowledge as well as general knowledge: the interpretation of the information in the text relies to a large extent on conceptual, causal, temporal and other assumptions which are not explicit in the text but which have to be encoded in a knowledge base for this text understanding system. Some of this is commonsense knowledge, for example, that people who are admitted to hospitals are "patients" or that patients must be dead before autopsies are carried out. And some of this is expert's knowledge, for example, knowledge about the possible positions of ventricles.

In this paper we describe the organization and contents of the knowledge base (KB) developed for the processing of the PDS letters. We mainly concentrate on "what to represent" rather than "how to represent", and on how the KB helps to interpret NL utterances rather than which inferences one can draw from these utterances. In our development we used a type-feature formalism CL-ONE (Erbach 1995) which allows for internal consistency of the KB. Our main interest, however, lies in the interaction of knowledge and natural language and we tackle this by populating the KB with integral lexico-conceptual knowledge structures. On the basis of some example texts and the system output, we will illustrate how these integrated knowledge structures are used in processing to yield the

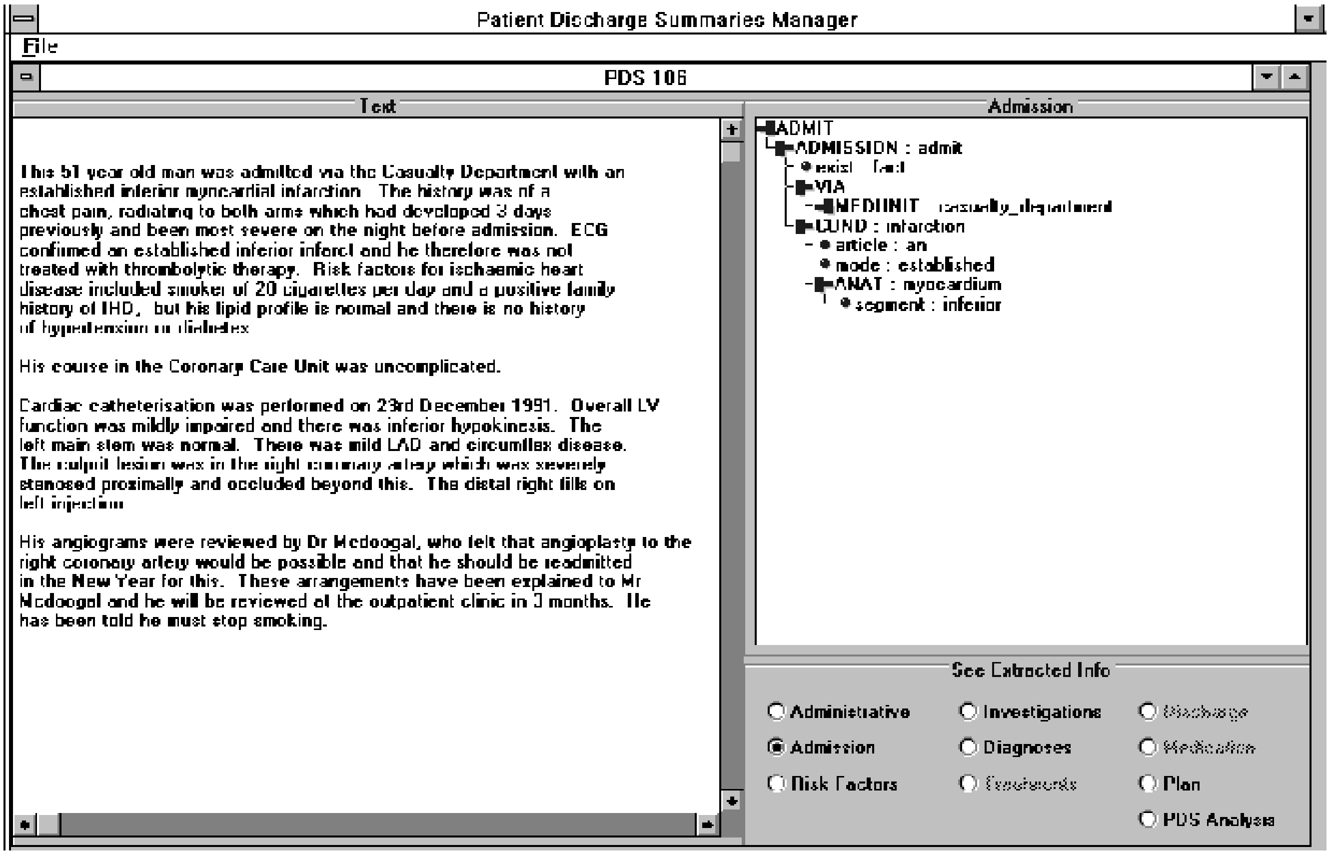

Figure 2: This figure shows the user interface of the system. In the left window the PDS text is represented. In the right bottom area the user can specify the kind of extracted information to display for verification and correction. In this case it is the information about the event of admission. The right top window displays the extracted fact about the patient admission: it really happened (fact), was via the casualty department, the reason was the condition of "established infarction" of the body-part "myocardium" in the inferior segment.

desired interpretations.

2 The Text Processing Paradigm

There are two main paradigms in parsing[1] of NL: syntax driven and semantics driven.

The syntax driven paradigm attempts to produce a syntactic tree of a sentence for further analysis. Although not only syntactic knowledge is used in that process the main target is to build a detailed syntactic representation before doing other stages of the processing. There have been developed quite a few syntactic parsers some based on Phrase Structure Grammars like GPSG, HPSG, etc. (e.g. Boguraev 1979), others on Transformational Grammar, as for example, PARSIFAL (Marcus 1980), others on ATN, for example, LUNAR of Woods (1970). Problems with the traditional syntactico-semantic approach to text processing are well known. The idea of incremental layered interpretation of the text into, first, syntactic, then semantic (logical forms) and, finally, pragmatic structures suffers from the enormous lexical and structural ambiguity of natural language. For example, the earlier version of our PDS understanding system was built in this way using Alvey's GPSG large coverage English Grammar (Grover *et al* 1993) and event-and-state knowledge base which helped the grammar to disambiguate words and attachments and also was a source for enriching the output representation with default values of implicit thematic roles (Whittemore *et al* 1994). Roughly speaking, for unrestricted texts only about 60 percent of sentences could be successfully syntactically parsed. Semantic information like selectional restrictions helped to raise the score but then the interpretation of the syntactic structures into logical forms (LFs) suffered from the same problems. LFs are meant to represent so-called "literal" meaning of an utterance. This meaning is composed from the meanings of the words and some extra linguistic information derived from the knowledge base. In many cases, however, the literal meaning is hard to capture while the pragmatics of the utterance can be quite clear. Problems with word polysemy, contextually and pragmatically dependent meanings, ill-formed but informative structures make the composition of LFs fragile and computationally expensive, leaving the final success rate at about 30 percent. Clearly, this is far from being acceptable in a real task environment where language technology is used as an instrument to accomplish a larger task as, for instance, population of a data base, control of a robot, etc. In summary, it doesn't look like the syntax-driven text understanding paradigm which is based on so-called "compositional interpretation" of utterances using syntactic and semantic phrasal structure rules is ready to tackle processing of unrestricted texts in a robust and efficient way.

The other major paradigm - semantic parsing - does not see the need in a separate syntactic analysis. There is a variety of approaches in this camp ranging from the use of quite detailed syntactic knowledge spread over semantic knowledge to parsing without syntax at all. Initially semantic parsers (e.g. Riesberg&Schank 1976, Wilks 1975) were

[1]Here we adopt the definition for parsing proposed in Wilks&Jones (1985) where it is defined as a computational process that takes sentences in natural language and converts them to some representational structure. Parsing is thus not restricted only to syntactic operations.

based on general semantic knowledge but later this was complimented with domain knowledge expressed in scripts, plans, theories, etc. in parsers like FRUMP (deJong 1987) or AD-HAC (Cater 1981). This approach proved to be more feasible to apply to real-world language processing. The main idea behind it is to go directly to the pragmatic interpretation without the composition of LFs. It relies on domain and task pragmatics tightly bound to the domain sublanguage rather than extensive linguistic knowledge and general syntactic and semantic interpretation of NL texts. Such systems are still aimed at the processing of unrestricted NL input, however, their output is rigourously specified, for instance, a set of parametrized conceptual schemata for the domain. The top-down filter on what is informative in the text in a sense shifts the complexity of the NL "understanding" from the linguistic to the knowledge level making the overall task more balanced. In fact, this reformulates the whole approach to text processing from "what can it be?" for a NL expression to "how can it be expressed?" for a conceptual schema. Thus, the system can be organized around conceptual structures of the domain rather than linguistic structures of the natural language. The notion of sublanguage is another fundamental assumption in the domain-bound approach. It provides a bottom-up constraint on the language use which comes from the nature of the NL corpus itself. The corpus usually represents a domain-oriented and in a sense restricted (Sager 1984) use of language where the range of linguistic structures is somehow naturally tailored and limited for the domain. This significantly simplifies the job for the linguistic description to deal with the finite number of established cases in the sublanguage rather than with infinite possibilities of the NL in general. This by no means contradicts the requirement for the NL input to be unrestricted since the sublanguage exists naturally and is described on the basis of empirical observations. The major drawback of the semantic parsers is that unlike the syntactic ones they have to be endowed with a vast body of knowledge which is mostly non-portable across domains.

The information extraction technology that matured at the ARPA-sponsored Message Understanding Conferences (MUC) provided a great contribution to the real-world parsing of NL. The experience gained in the MUC enterprise showed the major trends in the evolution of the NLP systems. In general, the standard rule-based compositional NLP methods have been augmented (if not replaced) with pattern-based fragment interpretation. Many researchers adopted finite-state sublanguage grammars which rely on domain semantic categories and phrasal patterns instead of combining different general knowledge sources such as syntax, semantics, phrasal rules, and domain knowledge at run-time. Jacobs *et al* (MUC-5 p.111) categorized MUC-5 systems into three distinctive groups. About half of the systems used linguistic analysis driven by "traditional" phrasal structure rules with elaborated syntactic and semantics components. The second group was populated by fragment parsing systems: they employ partial parsing by using a combination of syntactic and domain knowledge to guide the composition of syntactic chunks. They used heuristics and patterns for fragment combination, but their fragment parsing was based on deep syntactic and semantic analysis. And the third group was characterized as the "pattern matching" one. This term actually was used in a loose sense and was applied to all the systems which did not rely on compositionality of constituent structures but rather on a shallow construction-driven analysis and direct pragmatic interpretation of entire text fragments. The top performance at MUC-5 was achieved by two "pattern-matching" systems – SHOGUN/GE-CMU (Jacobs *et al* 1993), FASTUS/SRI (Appelt *et al* 1993) and one "fragment parsing" system – PLUM/BBN (Weischedel *et al* 1993). The SRI's MUC system is one of the examples of the complete re-orientation in the text processing paradigm. This system initially employed a full-power text-processing system TACITUS (Hobbs 1986) with modules of elaborated syntactic and semantic analysis and then the abductive reasoner for resolving general and "local" pragmatics. However, later on[2] that system was replaced by a "pattern-matching" FASTUS system (Appelt *et al* 1993) which initially was considered to be only a preprocessor for TACITUS. FASTUS achieved a better level of performance with only one month domain-specific development for MUC-4 task. The other best-scoring system - SHOGUN (Jacobs *et al* 1993) - also represents a successful shift from the traditional text processing paradigm into the finite-state direction. Thus, GE-NLPToolset (Krupka *et al* 1992) which was based on chart-style syntactic parser guided by conceptual expectation was replaced by the "pattern-matching" SHOGUN system in MUC-5 competition.

The approach taken in our development falls more or less into the category of "pattern-matching" systems: we don't have an explicit stage of syntactic analysis or semantic analysis, and directly interpret text fragments into the conceptual structures of the domain. During the analysis we clearly use both syntactic and semantic information, as well as a combination of compositional and direct interpretation rules both of which are based on domain and lexical knowledge. The main principles of our approach can be summarized as follows:

- We draw a clear distinction between KBs for natural language processing (NLP) and KBs for reasoning with knowledge. KBs for NLP should be directly aimed at helping the system to convert free text into the logical format, while a reasoner draws conceptual inferences which are not necessarily important for the text processing, but rather concerned with what implications the information captured from the text can have. For example, for the text processing purposes it is not necessarily important to infer that if somebody paid money for something this money changed its location. On the other hand, it might be crucial for the text processing to know who is referred to as "the buyer" and who is "the seller".

- Consequently, a KB for NLP should be explicitly targeted to the domain, as well as the information to be captured and the texts to be processed. In such KBs conceptual knowledge of the domain is closely tighted with linguistic structures from the underlying corpus. This requires a representation capable of capturing the correspondence of conceptual and linguistic structures.

- Conceptual knowledge in the KB is built and organized in a top-down fashion and explicitly targeted to represent information which is identified to be important. Linguistic knowledge is built around the conceptual structures in a bottom-up and empir-

[2] starting from MUC-4 competition

ically driven way and encodes the actual textual representations of particular classes of conceptual structures.

- Such organization requires rapid development of domain-oriented knowledge bases, rather than specialization of general-purpose knowledge bases to new domains. This by no means contradicts the idea of reusability of already acquired knowledge. Reusable parts can be modularized remaining empirically valid bottom-up observations.

- The KB is the major component in the processing of natural language. The major trend is to go directly to the real semantic contents without producing the literal meaning of an utterance and to use entire lexical constructions (e.g. Zadrozny *et al* 1994) rather than single words in composition of the conceptual representation of the text. General text processing techniques like syntactic analysis and semantic analysis, should be used only as back up facilities.

3 System Architecture

As it is stated in (Jacobs *et al* 1993) among the "pattern-matching" approaches "the main differentiator is not in the basic processing algorithms but in the way that knowledge ends up getting assigned to various system components". The PDS processing system is not an exception in this respect and can be roughly described as a variation of the "Generic Information Extraction Architecture" outlined in Hobbs (1993). The following pipe-line modules can be recognized:

- *text zoner.* The zoner separates the PDS's header from the body text and extracts structurally represented information from the header. Next, the body is decomposed into paragraphs using simple deterministic rules. One rule, for example, says that if there is an empty line and the first non-blank character is a capital letter - this is a paragraph boundary. Another rule says that if the length of a line is less than normal (this is calculated using special heuristics) and the next line starts with more than one blank followed by a capital letter - this is a paragraph boundary. The other activity which the zoner performs is the recognition of itemized text segments.

- *preprocessor.* The preprocessor consists from several modules applied one after another:

  - *phrasal recognizer* - it groups sequences of words into single text elements. There are several kinds of phrases in use:

    1. general phrasal entries such as "at least", "as well as", etc.,
    2. date, name, measure, location expression patters which handle phrases like "15th of Dec. 1993" or "approximately fifteen and a half gm.".
    3. domain entity thesaurus expressions, for example, "right coronary artery", "left anterior descending", etc.

These phrasal patterns then are converted into SGML (Goldfab 1990) elements with their semantic types and part of speech information. For example, the phrase like "isheaemic heart disease" is recognized by the domain thesaurus as a sentence element (SEL) of patient condition of subtype IHD and assigned with "common noun" (NN) part-of-speech information:

```
<SEL type=phrase sem=COND stype=IHD pos=NN>isheaemic heart disease<SEL>
```

- *tokenizer.* This module tokenizes text into elements like sentences, sentence elements, etc. It takes the phrasal elements identified earlier equal to words and produces SGML entries for the text. For example:

```
<S>
 <SEL type=word>This</SEL>
 <SEL type=word>showed</SEL>
 <SEL type=phrase sem=COND stype=IHD pos=NN>isheaemic heart disease</SEL>
 <SEL type=punct>.</SEL>
</S>
```

- POS *tagger.* We use part-of-speech (POS) probabilistic tagger which is essentially a C++ re-implementation of the Xerox tagger (e.g. Kupiec 1992). First, every sentence element (SEL) of the type "word" is looked up in the lexicon for the parts of speech it can take on. For coping with unknown to the lexicon words special word-POS guesser based on morphological properties of word prefixes and suffixes was developed (Mikheev 1996). Then the words are disambiguated using the trained on the domain sublanguage Hidden Markov Model. The tagger achieves 97% accuracy on known words and 89% on unknown. For example, the sentence quoted above would be tagged as:

```
<S>
 <SEL type=word pos=DT>This</SEL>                - determiner
 <SEL type=word pos=VBN>showed</SEL>             - verb/past form
 <SEL type=phrase sem=COND stype=IHD pos=NN>isheaemic heart disease</SEL> - no
 <SEL type=punct pos=SENT>.</SEL>                - end of sent.
</S>
```

- *lemmatizer.* Tagged words are provided with their main forms. For example, for the past verb "showed" its main form is the verb "show":

```
<SEL type=word pos=VBN lemma=show/VB>showed</SEL>
```

- *preparser.* Here we use a finite-state context-dependent parser which operates over SGML marked sentence elements (SEL) and produces a skeleton recognition of Noun

Groups (nouns with their pre-modifiers), Verb Groups (verbs with their auxiliaries and pre-modifying adverbs) and some other linguistic constructions without resolving their internal structure. Here is an example of a noun group :

```
<NG head=12>
 <SEL type=word pos=DT lemma=this/DT>This</SEL>
 <SEL type=word pos=CD>65</SEL>
 <SEL type=word pos=NN>year</SEL>
 <SEL type=word pos=JJ>old</SEL>
 <SEL type=word pos=NN id=12>lady</SEL>
</NG>
```

- *semantic parser*. This is the same parser which is used for the preparsing but at this stage it employs domain semantic types as well as POS and syntactic information of the sentence elements and produces interpretations of sentential fragments direct into conceptual structures. For example, a rule can state that if a body-part pre-modifies a condition in a noun group, that body part is the location of the condition.
- *coreference resolution*. The same domain entities as, for example, body parts, can be referred to in different sentences and quite often by different strings. Consider: "her left descending", "the LAD", "the occluded artery", etc., cab be references to the same body part of the patient. Special heuristics try to resolve the related references.
- *fragment combination*. Every semantically parsed fragment has its own semantic type as, for example, patient-condition proposition. Special heuristics try to relate fragments to each other using the domain schema.
- *template generation*. The produced conceptual structures are fitted in the predefined domain forms.

The design of the zoner, tagger, lemmatizer, preparser and other modules is out of the scop of the current paper. Here we will concentrate on the representation of domain lexico-conceptual knowledge which is used both for the thesaurus matching and the syntactic parsing. This knowledge is uniformedly represented by means of lexico-semantic constructions which are explained in details in the following sections.

4 Lexico-Semantic Constructions

The fundamental idea behind our approach is grouping of lexical knowledge around conceptual structures rather than conceptual characterization of lexical knowledge. Thus, we are not interested what a particular word or a phrase can mean but rather by which constructions a certain fact can be expressed. Therefore, we don't have a so-called semantic lexicon i.e. words with associated semantic meanings and a grammar for their combinations into structures, but rather our conceptual structures are associated with entire lexico-semantic constructions (LSCs) they can be expressed by. The main assumption behind LSCs is that every piece of information can be expressed by a finite number of classes of NL expressions. Lexico-semantic constructions (LSCs) cover these classes of NL expressions and are pre-packed with rules for deterministic interpretation of these expressions into conceptual schemata of the domain. Thus, whenever a lexico-semantic construction is recognized in the text, a corresponding conceptual schema is generated. If no LSCs have been recognized in a certain text segment, the system leaves it uninterpreted.

A LSC can be seen as a triple $< P, L, C >$ where P stands for a set of precondition rules, L stands for a set of surface lexical structures and C stands for the resulting parametrized conceptual structure. The *precondition rules* (P) are mainly used for resolving anaphoric and definite NP references but in general they can constrain the context in which a particular construction is valid. To do that the rules can access the discourse (D). Every entity in the discourse is represented as a type-feature structure, which allows for partial matching of antecedents. The rules therefore can look for a certain antecedent in the discourse, as well as other conditions as, for example, that another entity of certain type should not be mentioned after it, or that the matched entity occurred within a certain distance back from the current sentence. For example, the rule:

P: (D_i **== <!med-examination> & [1,3]**) & (D_j **!= <!med-treatment> if j > i**)

says, check that there is in the discourse an entity of type MED-EXAMINATION (D_i **== <!med-examination>**), which occurred within the last three sentences (**[1,3]**) and that after it no medical treatment procedures were mentioned (D_j **!= <!med-treatment> if j > i**).

The *surface lexical structures* (L) can be seen as context dependent phrase-structure rules expressed in terms of regular expressions, lists of regular expressions, syntactic and semantic features and conceptual structures. They allow for repetition ("*" and "+" modifiers) and optionality ("?" modifier) of constituents and can refer to elements introduced by the precondition rules. The constituents of the surface lexical structures are not necessarily ordered as sentential patterns, since in inflected languages, like Russian or German, case markers have more important role than ordering, but for illustration purposes we start with the pattern-oriented form, which in fact works well for English. For example, a structure:

L: **{"this", "it"} <VG head:={"show","reveal","diagnose"} ta:= {past, prpf}> <!med-condition>**

describes a construction of three elements: a list of disjuncted strings (**{"this", "it"}**), a set of disjuncted verb groups with the heads "show","reveal" and "diagnose" used in the past or present perfect form (**< VG head:={"show","reveal","diagnosed"} ta:= {past, prpf}>**), and a conceptual structure MED-CONDITION. This construction is capable of covering phrases like:

This showed an occlusion in his left trunk.

It has revealed severe stenosis.
This diagnosed pulmonary heart disease.

as long as "an occlusion in his left trunk", "severe stenosis" and "pulmonary heart disease" can be interpreted into the conceptual type of MED-CONDITION.

The *parametrized conceptual schema* (C) specifies how the matched lexical structure should be interpreted. We represent these schemata as type-feature structures which can use variables to refer to elements matched in the discourse (P) or in the lexical part (L). For example **<!test-result test:=**D_i **result:=$X status:=@fact>** represents a structure of type TEST-RESULT with features test, result and status which are assigned with certain values. Now we can put together the P, L and C components and represent a complete LSC as shown in figure 3.

P : **(D_i = <!med-examination> & [1,3]) & (D_j != <!med-treatment> if j > i)**

L : **{"this", "it"} <!VG head:={"show","reveal","diagnose"} ta:= {past, prpf}> $X:<!med-condition>**

C : **<!test-result test:=**D_i **result:=$X status:=@fact>**

Figure 3: **test-res-anaph** : a LSC for the anaphoric "test-result" constructions.

The resulting conceptual structure C in figure 3 is composed from the medical examination D_i matched in the discourse by the precondition rule (P) and the medical condition of a body part ($X) matched in the sentence. The status "fact" is assigned to the whole proposition. In the conceptual graph notation the resulting structure would look like:

[test-result: x42]
→(test)→[med-examination: D_i**]**
→(result)→[med-condition: $X]
→(attr)→[status: @fact]

LSCs introduce two very important notions: the notion of *contextual interpretation* and the notion of *local synonymy*. The *contextual interpretation* means that all elements of a construction are interpreted only within the context of this particular construction unless stated otherwise. This allows for overcoming the lexical and structural ambiguity of the surface structure. There is no need for word disambiguation because they are unambiguous within a certain construction. At the same time the same word can be distributed across multiple constructions and play different parts in them. Moreover, there is no longer need for establishing the "meaning" of a word since the "meaning" of a construction is not a direct function of the "meanings" of its constituents. Thus, words and phrases contribute to but do not compose the meaning of the overall construction. The structural ambiguity is tackled by the explicit expectation of certain kinds of structures described in the L patterns of the LSCs. So, for example, an L rule can explicitely prescribe where to attach a prepositional phrase of a certain type in a certain context. In a sense it acts similar to lexical preferences with the difference that the attachment is carried out in the resulting conceptual structure C rather than the lexical form L. This has an advantage to handle cases when lexically the attachment is unclear but pragmatically unambiguous. For example, in the phrase

Dr.Johns admitted Mr.McDoogal with severe myocardial infarction on Friday.

although it is unclear where the prepositional phrase "with severe myocardial infarction" should be attached to[3] pragmatically it unambiguously can be interpreted as the reason for admission.

Different parts of a construction can be satisfied by several lexical or structural items. These items are grouped together into lists of *local synonymy* which are valid only for a particular construction or a set of constructions. For instance, verb groups for "show", "reveal" and "diagnose" are local synonyms in the context of the "test-res-anaph" LSC shown in figure 3. Local synonymy allows for not only identification of words and phrases with common functions in a given context but also it allows the system to recover possible spelling errors and typos. For example, in general it is not true that a stand alone "o" is a misspelling of "of", however, in a position where "of" is expected we can put "o" in the list of local synonymy: **{"of", "o","f"}**.

Another important advantage of LSCs is that they can naturally incorporate multiple knowledge sources: domain types, general semantic categories, syntactic structures and features, phrases, words, etc. Thus, for instance, in the LSC represented in figure 3 a syntactic structure

<!VG head:={"show","reveal","diagnose"} ta:= {past, prpf}>

introduces a local synonymy of three verb groups. They are followed by a domain type: **<!med-condition>** which is also assembled only for the given construction. The first element of the whole construction is a list of words which are synonyms in the given context: **{"this", "it"}** .

An important property of LSCs is that they are empirically driven and close to the ground. They do not impose any constraints on well-formedness of the constructions and reflect only how it is usually said or written in real life. It is well known that many generalizations, which seem very plausible, usually are jeopardized by many counterexamples from the real-life language use or rather misuse. LSCs tackle this problem in a bottom-up fashion allowing for encapsulation of lexical and domain knowledge into specific cases. It doesn't mean that there are no generalizations over the constructions but such generalization are just the constructions which cover broader classes of lexical expression.

[3]one can find many arguments for the attachment to the verb "admitted" and as many counter-arguments for the attachment to the noun "Mr.McDoogal".

4.1 Local Synonymy Grouping

Lexical items grouped into the local synonymy lists are one of the major building blocks for LSCs. Some groups of local synonymy can be shared by many patterns, thus representing cross-contextual regularities. In order to allow for synonymy group sharing we introduce a device of macro substitution – operator **#define**:

#define ARTERY := {"artery","vessel","one"}
#define A-ART := {"an","a"}

#define WAS := {"was","appeared to be"}

Thus, one can refer, for instance, to the indefinite article list {"an","a"} as \$A-ART in different constructions. Let us consider a construction which would cover a class of utterances as:

Marginals was dominant.
Circumflex appeared to be a dominant vessel.
LAD was doinant artery.
Right coronary artery was a doinant one.

To handle these cases one can create a construction:

L: **\$X:<!art> \$WAS \$A-ART? "dom?inant" \$ARTERY?**

C: **<!med-condition-prop bodypart:=\$X cond:=\$X→dominancy:@yes status:=@fact>**

This construction says that if a conceptual type of artery is followed by a member of the list \$WAS (which are "was" and "appeared to be"), then followed optionally by an indefinite article, then by the word "dominant" with a possible misspelling and then optionally by one of the words from the local-synonymy group \$ARTERY[4], create an instance of the type MED-CONDITION-PROP (similar to one discussed in section 1) and assign the bodypart field with the artery (\$X), and the condition field with the feature "dominancy" of the artery with the value "yes". The whole proposition has the status "fact" in this case. A similar class of utterances as:

Marginals was bigger than normal.
Circumflex appeared to be a big vessel.
LAD was enlarged artery.
Right coronary artery was a large one.

can be handled by a similar construction which reuses the above introduced local synonymy groups:

#define ENLARG := {"big","bigger than normal","enlarged","large"}

L: **\$X:<!art> \$WAS \$A-ART? \$ENLARG \$ARTERY?**

[4]Note, that here we explicitely rule out the meaningless anaphora resolution for the word "one" - it is explicitely interpreted as \$ARTERY.

C: **<!med-condition body-part:=$X cond:=$X→size:@enlarged status:=@fact>**

Unlike semantic tagging, local synonymy groups and their macro-names are recognized only in a particular context imposed by a corresponding pattern. Thus, for example, there is no general assumption what the word, say, "big" can mean, but the rule above locally interprets it into the value "@enlarged" of the feature "size". Unlike content words, words which play the structural role in a particular utterance, like those from groups $A-ART (indefinite article), $WAS and $ARTERY are usually shared among many constructions. The groups given so far are empirically based and independent of any other constructions. So, if the indefinite article makes a different contribution in another context, this can be simply encoded as another group.

In many cases a better level of generalization can be captured by evaluating feature-value contents of a synonymy group on the fly. To be able to do that we introduce the function *eval* – contextual evaluator. This evaluator binds a list of phrases to a certain conceptual type, for example **@enlarged ←{"big","bigger than normal","enlarged","large"}** , and every time one of these phrases is encountered in a certain context, which is defined by the whole LSC, it is replaced by the corresponding concept. To provide an access to a particular feature of a type we use the arrow operator, so, the structure **$X→size:$Y** tells to assign the feature "size" of what is bound to the variable $X with what is bound to the variable $Y. Consider the example below:

#define ENLARG := {"big","bigger than normal","enlarged","large"}
#define ENSMALL := {"small","smaller than normal","smallish"}

#eval LRG-SML := @enlarged ←$ENLARG; @ensmalled ←$ENSMALL

L: **$X:<!art> $WAS $A-ART? $Y:$LRG-SML $ARTERY?**

C: **<!med-condition body-part:=$X cond:=$X→size:$Y status:=@fact>**

In the above construction the evaluator $LRG-SML is fired only if all other conditions are satisfied. It evaluates groups of expressions (words, phrases, regular expressions) into conceptual types: @enlarged or @ensmalled which are declared in the KB as possible values for the feature size of the type artery. Then the evaluated value is bound to the variable $Y and thus can be used in the conceptual structure *C* to assign a particular value to the feature size.

Using this LSC sentences like:

Right coronary artery appeared to be smaller than normal.
Right coronary artery was a *small.*

will be interpreted regardless of the grammatical irregularity in the second sentence into the structure:

<!med-condition-prop bodypart:=#rca cond:=#rca→size:@ensmalled status:=@fact>

or in a more perspicuous CG notation:

```
[@med-condition-prop: #25]
  →(condition)→[size:@ensmalled]←(attr)←[bodypart: #rca]
  →(bodypart)→[bodypart: #rca]
  →(attr)→[status:@fact]
```

4.2 Handling Sequences of Conjuncts

When a certain proposition is held over a set of objects or attributes they are introduced in the text as a sequences of conjuncts. For example:

Right coronary artery, LAD and circumflex were small arteries.

A straightforward capturing of such a construction in an LSC is:

```
#define WERE := {"were","appeared to be"}

L: $X1:<!art> ,? $X2:<!art> ,? and $X3:<!art> $WERE $Y:$LRG-SML ($ARTERY"s")?
C: <!med-condition cond:=$X1→size:$Y cond:=$X2→size:$Y cond:=$X3→size:$Y >
```

This is not very elegant because for a conjuncted sequence which consists of a different number of conjuncts we will have to explicitly introduce another but almost identical rule. Grouping conjuncts together in a single composite type before firing an LSC is also problematic because in some cases two objects of the same type with the conjunction in between, belong to different clauses as in the sentence:

There was an occlusion in the LAD and the circumflex was bigger than normal.

the conjunction holds between two propositions rather than the two arteries (LAD and circumflex). Thus, first we need to make sure that the last conjunction element is not the subject of the next clause and second, that all conjunction elements are of the same semantic type. To allow for any number of the elements in a conjuncted sequence we have to support typed and indexed variables to produce multiple structures.To make the notation more abstract and elegant we can introduce a special function *"set"*:

```
#set(<!obj>) :=
L: <syn-type=VG> $Xn(<!obj>):{<!obj> (,? <!obj>)* ,? and <!art>} <syn-type!=VG>
L: <syn-type!=VG> $Xn(<!obj>):{<!obj> (,? <!obj>)* ,? and <!obj>}
```

This rules forbid to fire the construction if the last element of the conjunction belongs to a different clause: so if the conjunction belongs to the preceeding verb group (VG) it cannot be immediately followed by another verb group. The indexed variable **$Xn(<!obj>)** has an explicit constraint for the conjuncted elements to be of the same type. Since the second component inside the curly brackets (**(,? <!obj>)***) is "star modified" i.e. by the standard convention can appear zero or more times, the number of indexed variables $X is greater or equal two, because the first and the last elements are obligatory. The indexing of the variable becomes implicit because the *set* function always requires an indexed variable.

Actually, variable indexing is supported both for the group reading and the distributed reading: the syntax **$X:set(<!obj>)** is interpreted into **$X1:<!obj>...$Xn:<!obj>** (the distributed reading), whereas the syntax **$X:(set(<!obj>))** requires to assemble a set object of the type "obj" , for example **<!art type:=set el:=rca ..el:=lad>**, which consists of all matched single arteries as its elements, and then apply all further transformations to this set. Then we can write a construction:

L: **$X:set(<!art>) $WERE $Y:$LRG-SML ($ARTERY"s")?**

C: **<!med-condition-prop cond:=$X→size:$Y >**

which assigns variables (**$X1,$X2..$Xn**) with the **<!obj>** entities from inside of the curly brackets and reproduces these entities in the *C* part of the LSC.

The *set* function above is defined for the type "object": it takes any type which is a subtype of the type object and checks for the specified construction: **object (,? object)* ,? and object**. This function can be overloaded with a *set* function defined for a more specific type, so if, for example, there is a peculiar way to represent conjuncted sequences for arteries, we can define a function **set(<!art>)** which overloads the more general function anytime the function *set* is called for the type <!art> or its subtypes. Getting back to the example we introduced in this section, now we can write a generalized LSC (see figure 4) which covers utterances about the size for both a single artery and a conjuncted sequence of arteries. This construction applies its L[5] rules until one of them is satisfied. First, it tries the conjunctive group of arteries and then the singular artery.

```
#define ARTERY := {"artery","vessel","one"}
#define A-ART := {"an","a"}
#define WAS-WERE := {"appeared to be"}
#define WAS := {"was"} + $WAS-WERE
#define WERE := {"were"} + $WAS-WERE
#define ENLARG := {"big","bigger than normal","enlarged","large"}
#define ENSMALL := {"small","smaller than normal", "smallish"}

#eval LRG-SML := enlarged ←$ENLARG; ensmalled ←ENSMALL

L: $X:set(<!art>) $WERE $Y:$LRG-SML ($ARTERY"s")?
L: $X:<!art> $WAS $A-ART? $Y:$LRG-SML $ARTERY?

C: <!med-condition-prop cond:=$X→size:$Y status:=@fact>
```

Figure 4: **set-art-size** : a construction which handles both singular and conjunctive constructions for the med-condition-prop proposition.

4.3 Handling Objects-Property Phrases

Direct lexicalization of object-property phrases as introduced earlier is a robust but not very elegant way of description. For instance, in the figure 4 we introduced an LSC which would

[5]since there is no *P* element.

cover phrases as: ARTERY WAS ENLARGED/ENSMALLED. One, however, can easily guess that almost identical structure will be appropriate to cover manifestations of other properties of arteries:

Right coronary artery, LAD and circumflex were **small** *arteries.*
Right coronary artery, LAD and circumflex were **dominant** *arteries.*

Circumflex appeared to be **enlarged**.
Circumflex appeared to be **non-dominant**.

To avoid the repetition of the identical structures we introduced a function called "prop". We don't provide a definition for this function since it has rather procedural flavor but describe in details how it works. The *prop* function takes a string and leftmost or rightmost evaluates whether it contains a phrase which corresponds to a value of a feature of a certain conceptual type, and if it does, the function retrieves the feature and assigns it with the value. First, a type with its compatible features, their possible values and lexical manifestations of these values should be introduced in the KB. For example, for the type artery we can write an entry as follows:

```
#define ENLARG := {"big","bigger than normal","enlarged","large"}
#define ENSMALL := {"small","smaller than normal"}

#define DOMINANT := {"dominant","doinant","doinat","dominat","doininant","co-dominat"}
#define NON-DOM := "non-? *"$DOMINANT

<!art
  size :
          enlarged →$ENLARG
          ensmalled →$ENSMALL
  dominancy :
          yes →$DOMINANT
          no →$NON-DOM
>
```

This knowledge base entry introduces some features of interest for the type artery. These features are linked with the phrasal structures which manifest values of these features in the text. Using the *prop* function now we can generalize the construction shown in the figure 4 for the whole class of properties for the artery:

```
L: $X:set(<!art>) $WERE $Y:prop(<!art>) ($ARTERY"s")?
L: $X:<!art> $WAS $A-ART? $Y:prop(<!art>) $ARTERY?
C: <!med-condition-prop cond:=$X→..:$Y status:=@fact>
```

This construction incorporates a reference to the information which is determined at run time by the *prop* function: the variable $Y is bound to a value of any property (feature) of the type artery and in the *C* part of the LSC the name of the feature itself is introduced as "..", which means "to be determined from $Y". This, actually, imposes certain restrictions – if the same phrase corresponds to two different values of the same or different features of the same type, the feature cannot be deterministically identified. In fact, this is not disappointing at all, since it appears to be natural to use different words to refer to different properties of the same type and this was confirmed in our empirical observations. Here is an example of an application of the generalized pattern introduced above to the sentence *"Circumflex appeared to be non-dominant."*:

- take the whole sentence and match the longest leftmost subpart into the type "artery" as stated in the L part of the LSC. This is going to be a single word "Circumflex" in our example and the variable **$X** is bound to **#circumflex**.
- take the remaining part of the sentence and match the longest leftmost subpart to one of the phrases from the $WAS local synonymy group. This is the phrase "appeared to be" in our example.
- take the remaining part of the sentence and match the longest leftmost subpart to one of the phrases introduced with the values of the artery type. These are groups $ENLARG, $ENSMALL, $DOMINANT and $NON-DOM and the matched string is "non-dominant" from the group $NON-DOM.
- retrieve the value and the feature associated with the last matched string. In our example this will be the feature "dominancy" with the value "no".
- assemble the structure C with the feature and its value, which in our case will be **<!med-condition-prop cond:=#circumflex→dominancy:@no status:=@fact>**.

The *prop* function determines the feature and its value for the type but doesn't deal with sequences of conjoint features, as in:

Circumflex appeared to be non-dominant and enlarged.

This can, however, be captured by applying the *set* function over the *prop* function:

#set(prop(<!obj>)) := {prop(<!obj>) (,? prop(<!obj>))* ,? {"and","as well as"} prop(<!obj>)}

Now, we can represent the generalized LSC as a knowledge cluster (see figure 5), which is able to cover phrases like:

Circumflex was an enlarged vessel.
Circumflex and LAD appeared to be dominant.
Circumflex appeared to be non-dominant and enlarged artery.
Circumflex, RCA and LAD were dominant and smaller than normal.

4.4 Unknown Word Recovery

Lexico-semantic constructions as represented in figure 5 provide an efficient and robust way of dealing with broad classes of lexical expressions. However, such constructions assume that all words in the sentence are covered by the KB entries. This is clearly an oversimplification of real life which makes the LSC extremely sensitive to unknown words

```
#define ENLARG := {"big","bigger than normal","enlarged","large"}
#define ENSMALL := {"small","smaller than normal","smallish"}

#define DOMINANT := {"dominant","doinant","doinat","dominat","doininant","co-dominat"}
#define NON-DOM := "non-? *"$DOMINANT

<!art
 size :
        enlarged →$ENLARG
        ensmalled →$ENSMALL
 dominancy :
        yes →$DOMINANT
        no →$NON-DOM >

#define ARTERY := {"artery","vessel","one"}
#define A-ART := {"an","a"}
#define WAS-WERE := {"appeared to be"}
#define WAS := {"was"} + $WAS-WERE
#define WERE := {"were"} + $WAS-WERE
#set(<!obj>) := {<!obj> (,? <!obj>)* ,? {"and","together with"} <!obj>}
#set(prop(<!obj>)) := {prop(<!obj>) (,? prop(<!obj>))* ,? {"and","as well as"} prop(<!obj>)}

L: $X:set(<!art>) $WERE $Y:set(prop(<!art>)) ($ARTERY"s")?
L: $X:set(<!art>) $WERE $Y:prop(<!art>) ($ARTERY"s")?
L: $X:<!art> $WAS $A-ART? $Y:set(prop(<!art>)) $ARTERY?
L: $X:<!art> $WAS $A-ART? $Y:prop(<!art>) $ARTERY?
C: <!med-condition-prop cond:=$X→..:$Y status:=fact>
```

Figure 5: **art-prop** : artery-property construction.

and phrases. So, if an unknown to the KB word occurs within a text segment which can be covered by a construction, the whole construction is going to fail. For example, a phrase like

RCA was large, occluded and nondominant.

will cease to assemble because the word "occluded" is not known to the system (see figure 5). This is very unfortunate, because the information about dominancy and size can not be captured as well. To provide a robust interpretation we must introduce a recovery procedure, which, for example, skips over all unknown and directly attached adjectives in the context of the *set* function:

```
#set(prop(<!obj>)) := prop(<!obj>) (,? {prop(<!obj>),<!adj>})* ,? {"and","as well as"} prop(<!obj>)
```

Here we explicitly state that instead of a registered property (**prop(<!obj>)**) an unknown word which is an adjective (**<!adj>**) can also appear in the list. Thus, the *set* function will allow the recovery category (in this case **<!adj>**) to appear instead of the **prop(<!obj>)** one, but the meaningless words will be skipped over, because the indexed variables of the *set* function are explicitly bound to the type **prop(<!obj>)** by the notation **#set(prop(<!obj>))**.

5 Interpretation of Multiword Terms

In this section we exemplify the application of the LSCs described above to the interpretation of multiword terms. Multiword terms are usually represented as compound noun groups and cover a sufficient part of the surface lexical constructions, for instance:

severe inferior wall infarct
left coronary artery
coronary bypass grafting operation

There are two ways the KB deals with such expressions. The first way is called the direct lexicalization. Many multiword terms in fact do not have a clear compositional structure and must be represented directly:

```
#define SURG := {"operation","surgery"}
#define BYPASS := "by-? *pass"
#define GRAFT := {"graft", "grafting"}
#define CORART := "coronary (artery)?"

L: $CORART? $BYPASS $GRAFT $SURG?
L: $CORART? $BYPASS $GRAFT? $SURG
L: $CORART? $BYPASS? $GRAFT $SURG

C: <!therapy type:=bypass>
```

This LSC covers all the ways a *coronary bypass grafting surgery* is referred to in the text:

coronary bypass grafting operation
coronary artery by-pass grafting
by pass grafting
coronary graft surgery

For the second class of terms, however, a structural description can be provided. In Finin 1986 an approach for structural interpretation of compound nominals were proposed. The LSC as follows is a natural implementation of this idea:

```
P: within NG
L: $Y:set(prop(<!obj>)) $X:<!obj>
C: $X ..:$Y
```

This construction is very similar to the one presented in figure 5, but it expects the properties of the object to pre-modify directly the phrase which refers to the main object and occur within the same Noun Group as it is stated in the precondition rule. This, in fact, changes the interpretation of the construction: instead of creating a proposition with the status "fact" this construction just assigns certain features of the object with certain values. So, for example, a phrase like *"big dominant artery"* will be interpreted into the structure **<!art size:=@enlarged dominancy:=@yes>**.

Another important feature of the noun group interpretation is the support of embedded modifiers. For example, in the term: *"severe left myocardial infarction"* "severe" modifies the disease (infarction) whereas "left" modifies the body part (myocardium). The explicit information from the KB helps the expectation-based interpretation:

```
<!infarct
    →: {"infarct/nn","infarction/nn"}
    is-a : !disease
    loc : !body-part
    degree :
        high →{"severe","heavy","high grade"}
        mid →{"moderate",}
        low →{"mild",} >

<!myocardium
    →: {"myocardium/nn","myocardial/adj"}
    is-a : !body-part
    position-rl :
        right →{"right"}
        left →{"left"} >
```

First, the system matches the word "infarction" and activates the KB entry <!infarct>. Now the `set(prop(<!obj>))` part of the construction expects references to a body-part or to the degree feature. It checks the remaining string and activates the KB entry <!myocardium> with a higher priority for the next attachment. The remaining substring again is right-most checked for the compatibility with the <!myocardium> and since the word "left" corresponds to the position-rl feature it is saturated. The next word fails to be saturated with the <!myocardium> structure (which then is deactivated) and is checked against the <!infarct> structure, thus assigning the degree feature with the value "high". Thus the final structure of the analysis is:

```
<!infarct
    loc := <!myocardium position-rl:=@left >
    degree:=@high >
```

An important advantage of this algorithm is that every time only a small portion of KB rules (20-50 rules) driven by the contextual expectations is activated. Instead of activating all selectional restrictions for constituent words and then filter out all incompatible pairs we in a sense activate only those restrictions (they are implicitly specified in the KB) which are compatible with already identified types. So, for example, there can be hundreds of other entries for the words like "severe", "left", etc. but only those which are stated in the KB entries for the types BODYPART and DISEASE will be activated.

6 The Knowledge Base

The KB developed for our system is a loose collection of modularized knowledge clusters which are based on a set of commonly observed ontological postulates:

1. Everything that can be quantified over, measured or referred back to is granted the concept status.

2. There are two different concept categories: discrete and quantitative. While discrete concepts are instantiated into individuals, quantitative concepts are instantiated into points in a dimension or into intervals.

3. There are three main types of concepts: *entities*, *properties* and *roles* (cf. Guarino 1991).

Entities are semantically rigid (they preserve their reference in every situative framework) and essentially independent (they can be referred to directly). Entities have the following subcategorization:

entity → **stative**
- → **object** - *discrete individual*
- → **group** - *aggregation of objects*
- → **substance** - *mass*

→ **durational**
- → **event** - *discrete individual*
- → **process** - *aggregation of events*
- → **state** - *mass*

→ **quantitative**
- → **point** - *discrete individual*
- → **path** - *aggregation of points*
- → **dimension** - *mass*

Constraints and relations for combination of these types together are similar to those from the S&C theory of Jackendoff (Jackendoff 1983, 1987).

Properties are concepts which can be instantiated as values from a corresponding set of values. They are semantically rigid and founded on entities (i.e. recognized only in the presence of some entity). As with entities we can distinguish two kinds of properties: discrete and quantitative (scalar). Discrete properties are instantiated into members of their value set. Scalar properties are instantiated into AMOUNTS. *Roles* are types which are semantically non-rigid. They are founded on rigid types. For example, the role type DOCTOR is founded on the rigid type HUMAN and exists only in the framework of professional relations. Finally, *relations* correspond to the organizational properties of concepts and map concepts into concepts. A set of structural relations as *is-a*, *attr*ibute, *val*ue and some others, provides the core engine for structural knowledge organization.

Objects are traditionally represented as a collection of compatible properties and components. Properties are represented as value sets (or scales) and are linked with types they can be applied to. Eventualities are represented as a structure of their thematic roles, locative, temporal and causative-inchoative relations. In our analysis of eventualities we follow Parsons (1990) in distinguishing two kinds of eventualities: events, which *culminate* in a certain time-point, and states, which *hold* over a certain period of time.

A typical knowledge cluster is represented in the figure 5. It consists of type-feature conceptual structures and lexico-semantic constructions. For the implementation we use a type-feature logic formalism CL-ONE (Erbach 1994) aligned with the lexical operations for handling regular expressions of lexico-semantic constructions. Each of these clusters can be seen as a finite-state transducer which is capable to identify in the text a certain class of expressions and convert them into a conceptual structure. However, these transducers are arranged in a top-down fashion which distinguishes our system from the majority of other transducer-based approaches (Hobbs 1993). Traditionally, transducers are arranged into a bottom-up pipeline where more specific structures are assembled first, which provides building blocks for more general structures. In our system a transducer can call a more specialized transducer when a general construction requires a more specific construction. For instance, the LSC shown in figure 3 requires its last element to be of type MED-CONDITION. When the first two elements of this construction are matched successfully it calls the transducer which deals with MED-CONDITIONs, to satisfy the whole construction. It is very important to notice that a more specific transducer is called from and not before a general one. This allows for robust disambiguation of words. For example, the phrase "bypass grafts" can stand both for an artifact which is inserted into an artery and for the operation (bypass surgery) of inserting this artifact. However, when a general construction expects a therapy operation in a certain place, it calls a knowledge cluster which handles therapy operations and the phrase "bypass grafts" is deterministically mapped into **<!therapy type:=bypass >**.

To make the performance of the system more efficient, transducers, which are known to produce unambiguous results, are run in a bottom-up pipeline at the preprocessing stage. For example, expressions like *"15th of November 1993", "last May", "March the 15"* can be deterministically interpreted as dates. However, although a a stand-alone *"1993"* is very likely to refer to the year, we leave it to be interpreted in the framework of a particular construction which would require a date in a certain place.

The modular organization of the knowledge base allows for reusability of knowledge clusters. Among the reusable modules we have:

- **Date Specialist**. This module recognizes and interprets expressions which refer to dates, for example:

  #define OF:= {"of","o","f"}
  #eval MONTH := january ←{"Jan.","January"}; ...; december ←{"Dec.", "December"};

  L: **$X:set($NUMBER{"st","th"}?) $OF? $Y:$MONTH $OF? $D-ART? $Z:$NUMBER**
  L: **$X:$NUMBER{"st","th"}? $OF? $Y:$MONTH $OF? $D-ART? $Z:$NUMBER**

  C: **<!date type:=dmy day:=$X month:=$Y year:=$Z >**

  This construction is able to cover phrases like:

  15th of January 1995
  15 Jan. f the 1995
  15 and 16th January o 1995

- **Measure Specialist**: This module recognizes and interprets expressions which refer to amounts, durations, length, etc. For example:

two years and a half

approximately three hundred pounds or so

between 24 and 30 mgs.

- **Frequency Specialist**: This module recognizes and interprets expressions which refer to a frequency of an event. For example:

  {*once,twice,three times...*} {*a,an,per,every,each*} {*hour,day,month,year*}

- **Yearage Specialist**: This module recognizes and interprets expressions which refer to the yearage. For example:

  in his late 40s

  70 year old

  the age of 50

  aged 55

- **Name and Title Specialist**: This module recognizes and interprets expressions like proper names and affiliations similar to what is described in Wolinski 1995. Here are some examples of expressions which are covered:

  Mr.Smith

  Sandra McDoogal

  Dr.Johnson, head of the cardiac department

For our medical domain a few specialized knowledge clusters have been developed as well. These clusters operate with medical terms and constructions which assemble these terms into facts about the patient. Here is some list of domain dependent modules:

- **List of Heart Components**: this list includes direct lexicalizations of names of different arteries, tissues, etc. as, for example:

  #define CORART := "coronary (artery)?"

  L: **circumflex $CORART?**

  C: **<!art type:=circumflex>**

- **Constructions of Heart Components**: these constructions deal with type-feature structures of heart components. For example, a phrase like:

  origin of the circumflex coronary artery

  is interpreted by the construction

#define SN := {"portion","segment","branch","part","quarter","third"}

#eval SEGMENT := @origin←{"origin"}; @mid←{"middle", "mid", "medial"}

L: **$Y:$SEGMENT $SN? $OF? $X:<!art>**

C: **$X segment:=$Y**

into the **<!art type:=circumflex segment:=@origin >** structure.

- **List of medical conditions**: this list includes direct lexicalizations of names of different diseases and disfunctions, for example:

  L: **ischaemiac (heart)? disease**

  L: **ischaemia**

  L: **IHD**

  C: **<!disease type:=ihd>**

- **Condition-Bodypart Constructions**: these constructions deal with expressions which refer to certain conditions and bodyparts. For example, a phrase like:

  high grade stenosis at the origin of the circumflex coronary artery

  is interpreted as shown in section 5 into the structure:

  <!stenosis

  loc := <!art type:=circumflex segment:=@origin >

  degree:=@high >

- **Lists of Medical Examinations and Therapy Procedures**: this list includes direct lexicalizations of names of different medical procedures, as presented in the section 5.

- **Examination and Therapy Constructions**: these constructions deal with expressions which refer to when and where certain procedures were applied to the patient and what were the results. An example of such a construction is represented in the figure 3.

- **List of Drug Names and Dosage Constructions**: these constructions deal with expressions like:

  His medication on transfer was: Inderal LA 80 mgs b.d., Aspirin EC 300 mgs mane, Isosorbide Mononitrate 40 mgs t.i.d., Nifedipine Retard 20 mgs b.d.

7 System Walkthrough

In this section we will present a fairly detailed description of knowledge based text processing implemented in our system for "understanding" PDS letters.

7.1 Sentence 1

This 70 year old man was admitted from the Happy Valley Hospital on 28th January 1991 for angina pectoris.
After the preprocessing stage the system produces representation as follows:

```
this <!yearage year:=70> man was admitted from <!medunit type:=hosp name:=happy_valley> on <!date
day:=28 month:=jan year:=1991> for <!disease type:=ap>
```

Now, the system applies different high level constructions and one of them is the ADMISSION construction:

```
L: $X:<!person> was admitted ["from $Y1:<!medunit>", "to $Y2:<!medunit>", "on $Y3:<!date>", "for
$Y4:<!disease>", "for $Y5:<!test>", "by $Y6:<!person>"]+
C: <!admit ptnt:=$X src:=$Y1 goal:=$Y2 culm:=$Y3 reas:=$Y4 purp:=$Y5 doct:=$Y6 status:=@fact>
```

This construction expects after the word "admitted" one or more constructions from the list in square brackets, and the first (i.e. subject) position must be of the type PERSON. One of the constructions for the type person can interpret **this <!yearage year:=70> man** into **<!person sex:=m age:=<!yearage year:=70>>** and thus the whole sentence saturates into the resulting structure which then is trivially translated into the CG notation:

```
[admit:x12]
   →(ptnt)→[person:#1]
                    →(char)→[yearage:@70*year]
                    →(char)→[sex:@m]
   →(src)→[hospital: "happy_valley" x3]
   →(culm)→[date:x4 ]
                    →(day)→[day-num:28]
                    →(month)→[month-num:1]
                    →(year)→[year-num:1991]
   →(reason)→[proposition:
                    [angina-pectoris:x5]
                       →(exps)→[person:#1]
                       →(hold)→[time-period:x7]→(end)→[date:x8]←(<=)→[date:*x4]
                    ]
  →(attr)→[status:@fact]
```

This structure logically represents the proposition that an episode of admission which culminated on a certain date (x4), from the source which was the Happy Valley Hospital. This structure also reconstructs an implicit information that the same person who was admitted suffered from the "angina pectoris" which *held* prior the admission.

7.2 Sentence 3

Selective coronary arteriography and left ventriculography were performed on 29.1.91.

After the preprocessing stage the system produces representation as follows:

selective <!test type:=a-graphy> and <!test type:=a-graphy bodypart:=#lv> were performed on 29.1.91.

Now, the system applies different high level constructions and one of them is the construction for medical procedures:

$X:set(<!med-proc>) $WERE {"performed","carried out"} ["on $Y1:<!date>", "{for,of,because of,because,as} $Y2:<!disease>", "to $Y3:<!bodypart>", "by $Y4:<!person>"]+

C: <!med-proc-prop event:=$X cul:=$Y1 reas:=$Y2 loc:=$Y3 agnt:=$Y4 status:=fact>

The event <!med-proc-prop> culminated on date $Y1, medical procedures can be applied to certain locations of the patient body because of the disease of this patient. Now the *set* function tries to assemble <!med-proc> concepts into a set. The <!test> type is a subtype of <!med-proc> and a certain construction saturates "selective" with the <!test type:=arteriography>. Left ventriculography has been directly interpreted into a test which is performed to the left ventricle and the *set* function binds the variable $X to both of the tests. Then the construction expects a reference to a date and by applying the constructions for dates interprets "29.1.91" into <!date day:=29 month:=jan year:=1991>. The resulting structure after it was translated into the CG notation looks as follows

```
[med-proc-prop:x22]
   →(char)→[test:arteriography x27]
                    →(attr)→[mode:@selective]
   →(char)→[test:arteriography x28]
                    →(loc)→[bodypart:#lv]
   →(culm)→[date:x24 ]
                    →(day)→[day-num:29]
                    →(month)→[month-num:1]
                    →(year)→[year-num:1991]
   →(attr)→[status:@fact]
```

So, the event of medical procedure consists of two subevents which culminated on the date x24.

7.3 Sentence 4

This revealed a severe left myocardial infarct and a high grade stenosis at the origin of the circumflex coronary artery.

After the preprocessing stage the system produces representation as follows:

this revealed a severe left myocardial <!disease type:=infarct> and a high grade <!disease type:=stenosis> at the origin of <!art type:=circumflex>.

Now, the system applies different high level constructions among which a construction similar to one shown in figure 3:

```
P : ( D_i == <!med-proc-prop> & [1,3] ) & ( D_j != <!therapy> if j > i )

L : {"this", "it"} <!VG head:={"show","reveal","diagnose"} ta:= {past, prpf}> $X:set(<!med-condition>)

C : <!test-result test:= D_i result:=$X status:=@fact>
```

The discourse variable D_i is matched to the concept **[@med-proc-prop: x22]** discussed above, then the LSC calls constructions for medical conditions which saturate infarction and circumflex with their attributes. The resulting structure looks as follows:

```
[test-result:x31]
   →(test)→[med-proc-prop:*x22]
   →(result)→[med-condition:infarction x35]
                      →(loc)→[bodypart:#myocardium]
                                 →(attr)→[segment:@left]
   →(result)→[med-condition:stenosis x38]
                      →(attr)→[degree:@high]
                      →(loc)→[bodypart:#circumflex]
                                 →(attr)→[segment:@origin]
  →(attr)→[status:@fact]
```

8 Evaluation

Not done yet.

9 Conclusion

In this paper we described the organization and contents of the knowledge base (KB) developed for the processing of patient discharge summaries (PDSs)—letters sent by a hospital consultant to a patient's own doctor. This KB is the major component in the processing of natural language by our system and includes both the conceptual and lexical knowledge represented in a uniform way. Conceptual knowledge in the KB is built and organized in a top-down fashion and explicitly targeted to represent information which was identified by domain experts to be important. Linguistic knowledge is built around the conceptual structures in a bottom-up and empirically driven way and encode the actual textual representations of particular classes of conceptual structures. The fundamental idea behind this approach is grouping of lexical knowledge around conceptual structures by means of lexico-semantic constructions (LSCs), which cover entire classes of NL expressions and are prepacked with rules for deterministic interpretation of these expressions into conceptual schemata of the domain. Text processing with LSCs has the same spirit as the syntactic processing with the Constraint Grammar described in Karlsson (1990). The rules of LSCs constraint what is possible and expected if a given theory about the sentence is correct. Thus the rules discard as many alternatives as possible bringing as much information as possible as early as possible. Two important properties of LSCs - contextual interpretation and local sysnonymy allow for a distributed representation of lexical knowledge which doesn't require unnecessary generalizations.

Success of knowledge intensive text processing methods such as that advocated in this paper to a large extent depends on the feasibility of acquisition of this knowledge within a reasonable amount of effort. Recently quite a few methods for automatic and semi-automatic extraction of domain regularities from the underlying corpus were proposed. In Mikheev&Finch (1995) a workbench for the domain knowledge acquisition is described in some details. Using this workbench the actual knowledge engineering for our medical domain took us about two month. This allows us to suggest that although the KB developed for the PDS understanding task is explicitly targeted to the medical domain and its sublanguage, the approach advocated in this paper can be applied to multi-domain NLP. This will require a parallel development of lexico-semantic constructions for capturing information from the text and conceptual structures to accommodate this information.

10 Acknowledgements

This work was supported in part by the EC-funded ET-12 project "Methodologies for Constructing Knowledge Bases for Natural Language Processing Systems" and by the EC-funded AIM Project A2023 "An Access System for Medical Records using Natural Language".

References

Appelt, D. E., J. R. Hobbs, J. Bear, D. Israel, M. Kameyama and M. Tyson (1993) SRI: Description of the JV-FASTUS System Used for MUC-5. In *Fifth Message Understanding Conference (MUC-5).*

Boguraev, B. K. (1979) Automatic resolution of linguistic ambiguities. Technical Report 11, Computer Laboratory, University of Cambridge.

Brachman, R., D. McGuinness, P. Patel-Schneider, L. A. Resnick and A. Borgida (1991) Living with classic: When and how to use a KL-ONE–like language. In J. Sowa, ed., *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pp. 401–456. Los Altos, Ca.: Morgan Kaufmann.

Cater, A. W. S. (1981) Analysis and inference for English. Technical Report 19, Computer Laboratory, University of Cambridge.

Cutting, D., J. Kupiec, J. Pedersen and P. Sibun (1992) A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pp. 133–140.

DeJong, G. (1987) An overview of the FRUMP system. In W. G. Lehnert and M. H. Ringle, eds., *Strategies for Natural Language Processing*, pp. 149–176. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Erbach, G., S. Manandhar and S. Wojciech (1995) CL-ONE Users Manual. In preparation.

Finin, T. (1986) Constraining the interpretation of nominal compounds in a limited context. In R. Grishman and R. Kittredge, eds., *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*, pp. 163–173. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Goldfarb, C. F. (1990) *The SGML Handbook*. Oxford: Clarendon Press.

Group, T. P L U M. S. (1993) BBN: Description of the PLUM System as Used for MUC-5. In *Fifth Message Understanding Conference (MUC-5)*, pp. 93–107.

Grover, C., J. Carroll and T. Briscoe (1993) The Alvey natural language tools grammar (4th release). Technical Report 284, Computer Laboratory, University of Cambridge.

Guarino, N. (1991) A non-standard approach to terminological knowledge: the ITL system. In B. Nebel, C. Peltason and K. von Luck, eds., *International Workshop on Terminological Logics*, pp. 47–50, IBM Germany Research Center.

Hobbs, J. R. (1993) The Generic Information Extraction System. In *Fifth Message Understanding Conference (MUC-5)*, pp. 87–91.

Hobbs, J. (1986a) Sublanguage and knowledge. In R. Grishman and R. Kittredge, eds., *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*, pp. 53–68. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Hobbs, J. R. (1986b) Overview of the TACITUS Project. *Computational Linguistics* **12**(3).

Jackendoff, R. (1987) The Status of Thematic Relations in Linguistic Theory. *Linguistic Inquiry* **18**, 369–411.

Jackendoff, R. (1983) *Semantics and Cognition*. Cambridge, Mass.: MIT Press.

Jacobs, P. S., G. Krupka, L. Rau, M. Mauldin, T. Mitamura, T. Kitani, I. S. I and L. Childs (1993) GE-CMU: Description of the SHOGUN system used for MUC-5. In *Fifth Message Understanding Conference (MUC-5)*, pp. 109–120.

Karlsson, F. (1990) Constraint Grammar as a Framework for Parsing Running Text. In H. Karlgren, ed., *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics*, Vol. 3, pp. 168–173. University of Helsinki.

Marcus, M. P. (1980) *A Theory of Syntactic Recognition for Natural Language*. Cambridge, Mass.: MIT Press.

McDonald, D. (1994) Simplifying Analyses by Fitting the Semantic Representation Closely to Language. In *Knowledge Representation for Natural Language Processing in Implemented Systems AAAI-94 Fall Symposium*, pp. 90–96.

Mikheev, A. and S. Finch (1995) A Workbench for Acquisition of Ontological Knowledge from Natural Language. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 194–201, University College.

Onyshkevych, B. (1993) Template design for Information Extraction. In *MUC5*, pp. 19–23.

Parsons, T. (1990) *Events in the Semantics of English*. Cambridge, Mass.: MIT Press.

Reisbeck, C. K. and R. C. Schank (1976) Comprehension by Computer: Expectation-Based Analysis of Sentences in Context. Research Report 78, Department of Computer Science, Yale University.

Sager, N. (1984) Sublanguage: Linguistic Phenomenon, Computational Tool. Paper presented at the Workshop on Sublanguage, New York University, January..

Sowa, J. F. (1984) *Conceptual Structures: Information Processing in Mind and Machine.* Reading, Mass.: Addison-Wesley.

Spark Jones, K. and Y. Wilks, eds. (1983) Chichester: Ellis Horwood.

Whittemore, G. and J. Hicks (1994) ESK: Event and state knowledge base. In *AAAI Fall Symposium.*

Wilks, Y. (1975) An Intelligent Analyzer and Understander of English. *Communications of the ACM* **18**(5), 264–274.

Wolinski, F., F. Vichot and B. Dillet (1995) Automatic Processing of Proper Names in Texts. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 23–30, University College.

Woods, W. A., R. M. Kaplan and B. L. Nash-Webber (1972) The Lunar Sciences Natural Language Information System: Final Report. BBN Report 2378, Bolt, Beranek and Newman Inc., Cambridge, Mass.

Zadrozny, W., M. Szummer, S. Jarecki, D. E. Johnson and L. Morgenstern (1994) NL understanding with a grammar of constructions. In *Proceedings of the 15th International Conference on Computational Linguistics*, pp. 1289–1293.

Zadrozny, W. (1995) Parsing with Constructions. In *Knowledge Representation for Natural Language Processing in Implemented Systems AAAI-94 Fall Symposium*, pp. 147–155.