TUHH
Hamburg University of Technology

Institute for Software Systems

Evaluation of Automatically Generated Semantic Descriptions of Multimedia Documents

Diploma thesis by Kamil Sokolski

Hamburg, 29th January 2009

1st Supervisor: Prof. Dr. rer.-nat. habil. Ralf Möller
2nd Supervisor: Prof. Dr.-Ing. Dr. habil. Karl-Heinz Zimmermann
Advisor: M. Sc. Atila Kaya

Declaration

I hereby confirm that I have authored this thesis independently and without use of others than the indicated resources. All passages taken out of publications or other sources are marked as such.

Hamburg, 29th January 2009

City, Date Sign

Acknowledgements

I sincerely would like to thank Prof. Dr. Ralf Möller for fully integrating me into the BOEMIE project and providing me with this research work. Special thanks to Atila Kaya and Sofia Espinosa for their support and feedback. I also would like to thank Dr. Michael Wessel, who implemented the compute-abox-difference function in RacerPro, for his excellent RacerPro support and for explaining the main ideas of MiniLisp and abduction to me.

Contents

1 Introduction

The Internet is constantly expanding. Since its initiation, it has been growing impressivelyin size. In March 2000, there were one billion web pages indexed by search engines [1]. In January 2005, already 11.5 billion web pages were indexed [2]. Nowadays, in October 2008, there are at least 50 billion web pages indexed [3] [1] [2].

But not only is the amount of indexed web pages increasing, the kind of content also has changed over the last years. In the beginning of the world wide web, most of the web pages contained texts only. Nowadays, the rate of other media like image, video and audio is increasing rapidly. There are even websites such as *youtube* and *flickr*, which have specialized on providing the user with a specific kind of media documents, such as videos and images respectively.

To categorize text documents some chosen words within the text, so-called *keywords*, can be taken to represent the whole document. These keywords are then used to index repositories of documents.

Search engines allow for searching documents in the web with respect to. keywords used for indexing. To search for a document, a user has to guess keywords that he assumes are related to the document he wants and state this set of keywords to the search engine. The search engine looks up the index which documents contain the keywords. As a result, all documents that are indexed by these keywords are presented in the form of hyperlinks.

Since most images, video and audio documents do not contain text in digital form itself (except embedded text), this approach cannot be applied to them.

'Commercial search-engines often solely rely on the text clues of the pages in which images are embedded to rank images, and often entirely ignore the content of the images themselves as a ranking signal.'[4]

[1] Sorted on Yahoo!, Google, Windows Live Search and Ask

[2] The actual size of the Internet is approximately 500 times bigger then the indexed web pages because not every web page can be indexed by a search engine [1]

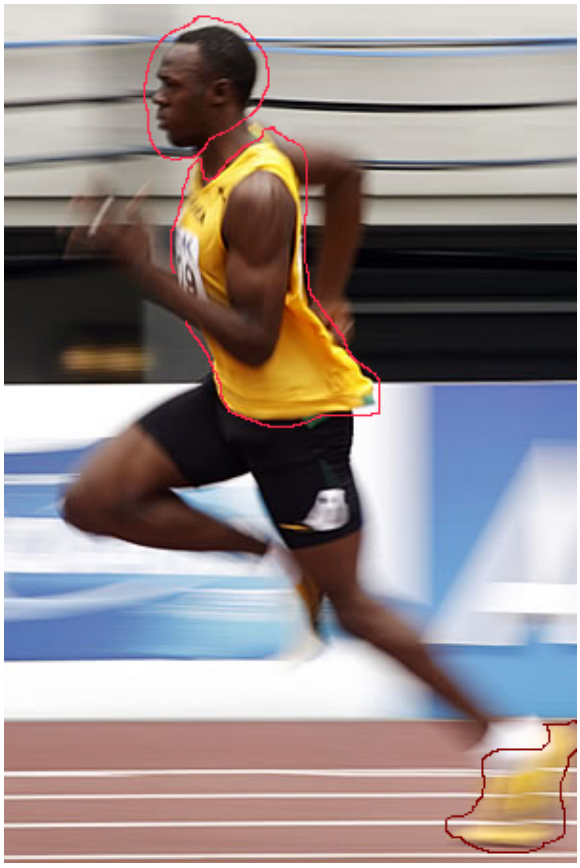

Figure 1.1: Image of Usain Bolt

Search engines do a good job when it comes to find a website that contains some information the user searches for. But content specific services (like semantic navigation or content specific advertisement) require rich descriptions of the content. These rich descriptions can be realized through so-called semantic descriptions, which not only provide a meaningful structured vocabulary for objects but also allow to relate objects with each other.

As an example consider the image of the 100m and 200m world record holder Usain Bolt in Figure 1.1. If for the image in Figure 1.1 semantic descriptions are available in terms of visible objects, relations among objects and corresponding regions in the image depicting these objects, this information can be used to offer a new form of interactions to the user. In this case, the user can interact with regions of the image to e.g. display a menu with a link to his biography, by clicking on the face of Usain Bolt, navigate to a web page about his running performance, by clicking on his feet or display a shoe advertisement by holding the mouse over the marked region of his feet.

A drawback of semantic descriptions as metadata is the big expense that is needed to produce them, if they are created manually. On a wide scale, only an automatic generation of semantic descriptions can improve the retrieval of multimedia documents and make content specific services affordable.

The EU-founded BOEMIE (Bootstrapping Ontology Evolution with Multimedia Information Extraction) project aims to develop methods for (semi-) automatic generation of semantic descriptions. BOEMIE follows a declarative approach where background knowledge directly affects the generated descriptions. In general, there exists several ways to model the domain/ background knowledge, and thus several alternative semantic descriptions can be generated for the same document.

This thesis investigates how automatically generated semantic descriptions can be evaluated using manually annotated descriptions.

In the next section we motivate this work in the context of the BOEMIE project.

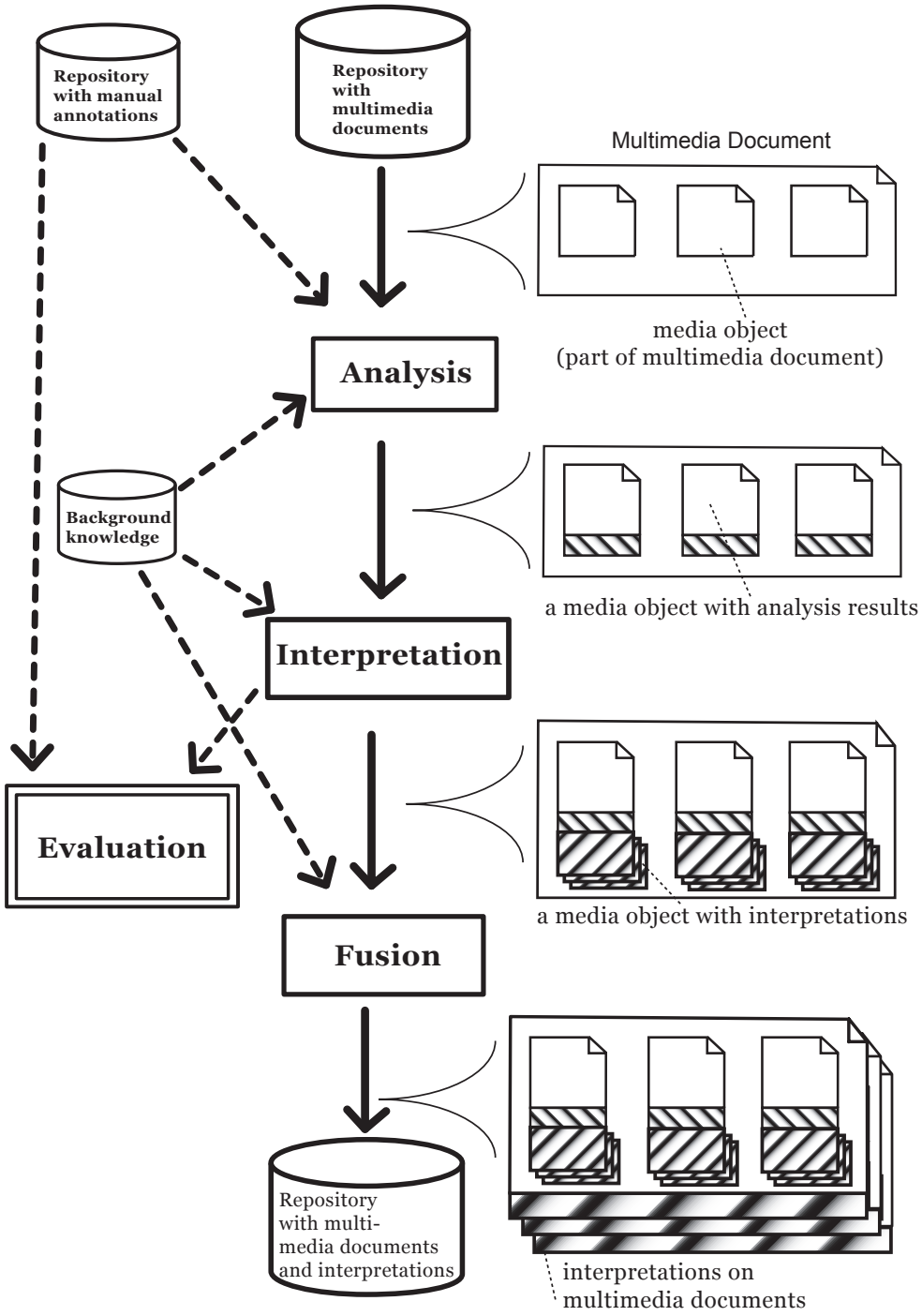

Figure 1.2: Schematic overview of the BOEMIE semantics extraction process

1.1 Motivation

To motivate the evaluation of system generated semantic descriptions we provide a brief overview of the 'BOEMIE semantics extraction process'. Figure 1.2 illustrates the BOEMIE semantics extraction process.

In the BOEMIE project a process called 'semantics extraction' is used to generates rich semantic descriptions for web pages that contain both textual and visual information, and audiovisual clips with audio and textual information(so-called videoOCR[3]). The semantics extraction process consists of three subprocesses, namely analysis, interpretation and fusion.

The *analysis* process extracts objects based on low-level features that can be identified directly within the media object. For example in the text modality the analysis process extracts names of persons, countries, cities, sports names etc; in the image modality, analysis can identify objects like faces of persons and bodies of persons and horizontal bars.

A multimedia document consists of some media objects, as shown on the right side in Figure 1.2. The techniques that are used in analysis differ depending on the modality of the media object. Therefore, analysis is done separately for every media object by using techniques such as supervised machine learning for image, audio and video media objects, and natural language processing for text. To extract objects and relations reliably, all these techniques require manually annotated descriptions as training data.

Besides training data, also domain-specific background knowledge is also required for the analysis process in order to create semantic descriptions. Informally speaking, domain-specific background knowledge defines the terminology of the domain of interest. It defines types in the domain and the relationships between objects. In BOEMIE the domain is about Athletics and the background knowledge contains names for concepts like pole vault competition or 100m run which themselves are related to other objects like an athlete with a role called 'hasParticipant'.

The analysis process produces already content specific descriptions but they are insufficient for services that need richer semantic descriptions which include abstract descriptions.

[3]OCR stands for **O**ptical **C**haracter **R**ecognition, which is an electronic translation of images of text into machine-editable text.

The next subprocess, namely interpretation, interprets analysis results by providing explanations, abstract semantic descriptions, for analysis results. The background knowledge includes so-called interpretation rules in order to constrain the space of possible explanations.

If analysis extracts some objects, the interpretation process tries to explain the existence and specific constellation of these objects. For example let us assume that in the athletics domain, analysis extracts a person, a pole and a horizontal bar in an image. With the background knowledge and interpretation rules for various sports competitions the interpretation process comes to the conclusion that the best semantics descriptions are these that describe the image as showing a pole vault scene.

The interpretation process is a modality-specific process, where analysis result of a media object is interpreted w.r.t. a single modality through rules designed for this specific modality.

In the last step of the semantics extraction process all interpretations of the media objects are 'fused' together. That is because a multimedia document fusion tries to relate the interpretation results from one modality with interpretation results of another modality. The idea here is that some information are represented redundantly in multiple modalities. In this way fusion creates interpretations for the whole multimedia document.

Example: Assume that an image has been interpreted as showing a person, whereas the corresponding caption has been interpreted as showing a person with a specific name. Most likely the persons interpreted in both media refer to the same person. Therefore fusion considers such interpretations in different modalities as interpretations of one and the same object.

In the BOEMIE project a logic-based approach to multimedia interpretation has been developed. In this approach the background knowledge does not only define the terminology of semantic descriptions but also acts as a declarative specification. This means that the background knowledge directly affects the automatically generated semantic descriptions.

To help improve the results of the semantic extraction process, we need to have the ability to compare rich semantic descriptions which haven been generated automatically using different background knowledge.

As a 'ground truth' for a good result, we take the manually annotated descriptions of media objects that are produced for training the analysis process. These manual annotations contain both annotations for objects of a lower abstraction level and

annotations for objects of a higher abstraction level.

In this work, evaluation is only done for the interpretation results in the text modality.

We define an automatically generated description as having a perfect quality if all objects (and relations) in the manually-annotation descriptions can also be found in the automatically generated metadata.

If we succeed to express the quality in terms of a value then we can compare results created with different background knowledge and therefore determine which background knowledge is more appropriate for automatic generation of semantic descriptions of multimedia documents.

1.2 Objective

When comparing two sets of semantic descriptions the key problem to be solved is to find a way to match (the descriptions of) objects (and their relations) from the manually annotated descriptions to automatically generated descriptions w.r.t semantics defined in the background knowledge. This cannot be based solely on syntax for two reasons:

1. The IDs of the objects do not match to each other as the human annotators define the names of objects during the annotation phase due to a convention and the system generates the names based on a unique name generation algorithm.

2. The information about an object may differ in the syntax but yet be similar on a semantically level due to the semantics stated in the background knowledge. E.g. a human annotator identifies the object in the image shown in Figure 1.1 as an athlete but the semantics extraction system interprets the objects as a runner. Syntactically, the words 'athlete' and 'runner' are completely different. However, according to the background knowledge a runner is also an athlete.

Therefore the development of a technique for semantically matching the manually and automatically generated descriptions with respect to the domain-specific background knowledge is the centerpiece of this work.

2 Introduction to Description Logics and DL Systems

The BOEMIE project uses the web ontology language (OWL) as a standardized way to represent semantic descriptions [5]. OWL is widely used in the semantic web context and is based on description logics. Description logics (DL) is a formal language with a well-defined semantic [6].

In this chapter, we introduce description logics (DL) as a suitable formalism for the representation of semantic descriptions. The usage of DL enables to build applications with standard reasoning services. We present some of the core reasoning services that are important for this work.

2.1 Description Logics

DLs is a family of Knowledge Representation formalism with different expressivity. In this section we introduce the syntax and the semantics of description logics $\mathcal{ALCQ}$ (Attributive Language with full Complement and Qualified number restrictions) as an example.

2.1.1 TBox

The TBox contains general knowledge (also called intentional knowledge) about the domain in the form of a terminology i.e. the vocabulary that is used. The vocabulary consists of concept and role definitions. A concept denotes sets of individuals and a role denotes a binary relationship between individuals [6]. For example, in the athletics domain *SportsCompetition* and *Athlete* might be concepts and *hasParticipant* might be a role between the concept *SportsCompetition* and *Athlete*.

Such elementary descriptions are called 'atomic concepts' and 'atomic roles' whereas

'complexed concepts' and 'complexed roles' can be built by using logical combinations of atomic concepts and roles. Let A be an atomic concept and R an atomic role, then the complex concepts C and D can be built using the following grammar:

C, D	$\longrightarrow$	A	atomic concept
		$C \sqcap D$	conjunction
		$C \sqcup D$	disjunction
		$\neg C$	negation
		$\exists R.C$	existential restriction
		$\forall R.C$	value restriction
		$\exists_{\leq n} R.C$	qualified minimum restriction
		$\exists_{\geq n} R.C$	qualified maximum restriction
		$\top$	universal concept
		$\bot$	bottom concept

The concept top $\top$ is an abbreviation for $A \sqcup \neg A$ and is always satisfiable. The concept bottom $\bot$ is an abbreviation for $A \sqcap \neg A$ and is always unsatisfiable.

With these kind of constructors, new symbolic names can be defined to introduce new complex concepts which can be stated as *terminological axioms* in the TBox. The expressivity of the language derives form the expressivity of its constructors [6].

As an example for a complex concept we could state

$$Runner \sqsubseteq Athlete \sqcap \exists participatesIn.RunningCompetition$$

In general, terminological axioms have the form of so-called *generalized concept inclusion* (GCI), $C \sqsubseteq D$ ($R \sqsubseteq S$), where C and D are concepts (R and S are roles). Also, axioms of the form of *equalities* $C \equiv D$ can be stated in the TBox, but these can also be expressed by two GCIs: $C \sqsubseteq D$ and $D \sqsubseteq C$.

To give a definition of the *semantics* for terminological descriptions, we consider *Interpretations* $\mathcal{I}$. An Interpretation $\mathcal{I}=(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a tuple of a non-empty set $\Delta^{\mathcal{I}}$, the domain of individuals, and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to every atomic concept description A and a set $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to every atomic role R [6]. Informally speaking the interpretation function maps form descriptions to objects of the world.

For complex concepts the interpretation function can be stated as follows:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \varnothing \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y. \text{ if } (x,y) \in R^{\mathcal{I}} \text{ then } y \in C^{\mathcal{I}}\} \\
(\exists_{\leq n} R.C)^{\mathcal{I}} &= \{x \mid \sharp\{y \mid (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\} \\
(\exists_{\geq n} R.C)^{\mathcal{I}} &= \{x \mid \sharp\{y \mid (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}
\end{aligned}
$$

2.1.2 ABox

The other part of the knowledge base, the ABox, describes a specific state of a world. In the ABox one can introduce individuals by giving them names and introduce properties of these individuals. For the properties one can make two kinds of assertions so-called *concept assertions* and *role assertions*. E.g. Using the concept C and the Role R one could state the following assertions for the individuals a and b: $C(a)$ and $R(a,b)$. The first assertion means that a is in the interpretation of C and the second assertion means that b is the filler of the role R for a.

To stick to our example of the world record holder Usain Bolt, $runner_1$ could be a name for the individual in the ABox that represents Usain. $Athlete(runner_1)$ would therefore state a concept assertion for the individual representing Usain Bolt. An other individual in the ABox could be $personName_1$ which represents the Name of Usain. The assertion $PersonName(personName_1)$ would state that $personName_1$ is actually an instance of $PersonName$. With the role assertion

$$hasNameValue(personName_1, `UsainBolt')$$

we introduce the so-called datatype filler, which allows us to attach values, like string or integer values to individuals. Finally, to state that the individual representing Usain Bolt has a name called 'Usain Bolt' we add the following assertion to our ABox $hasPersonName(runner_1, personName_1)$.

Semantics for an ABox can be defined by extending interpretations to individual names. In addition to the mapping of atomic concepts and roles to sets and relations, an Interpretation now also maps each individual name a to an element $a^I \in \Delta^I$ [6].

A concept assertion $C(a)$ is satisfied by the Interpretation $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Similarly a role assertion $R(a,b)$ is satisfied if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. If an interpretation satisfies each assertion in the ABox, it satisfies the whole ABox. The interpretation satisfies an

assertion of an ABox *with respect to* a TBox if the model of the assertion is also a model of the TBox [7].

Thus, a model of an ABox and a TBox is an abstraction of a concrete world where the concepts are interpreted as subsets of the domain as required by the TBox and where the membership of the individuals to concepts and their relationships with one another (in terms of roles) respect the assertions in the ABox[6].

2.2 DL systems

Analogous to a database system a DL system or reasoner is a system where information described in the form of description logics can be stored, organized and retrieved. The information described with description logics is stored in a knowledge base (also called ontology) which consists of two components, a 'TBox' and an 'ABox'.

The information described in the TBox is 'timeless' whereas the information in the ABox changes occasionally or even constantly. Compared to relational databases one can find some similarities between the TBox and a database schema and also the instances of a database and an ABox seem to be similar. Nevertheless, one main difference is that relational databases work on a *Closed World Assumption* (CWA) whereas DL-reasoners work on a *Open World Assumption* (OWA). CWA presupposes that the information about the domain is complete whereas OWA presupposes that the information about the domain is incomplete. For the retrieval of information Closed World Assumption means that anything which is not known to be true, is assumed to be false. Under Open World Assumption things that are not known to be true are not assumed to be false. '...this is one of the reasons why inferences in DLs are often more complex than query answering in databases.'[6]

To enable solutions for different reasoning tasks on problems described in DLs, DL-reasoner implement a set of algorithms for standard inference services, retrieval inference services and non-standard inference services. We present some of the core services that are important for our work.

2.2.1 Standard Inference Services

To check the *consistency* of an ABox w.r.t. the TBox a DL-reasoner has to test if all models of assertions in the ABox are also models of the TBox. All ABoxes that are

not consistent are called *inconsistent*.

To build a hierarchy of concept names the DL-reasoner has to check if a concept D *subsumes* a concept C w.r.t. a TBox. This is the case if, for all interpretations that are models of the TBox, $C^I \subseteq D^I$ holds [7].

The *classification* service finds all *children* and *parents* of a given concept C. The children of a concept C are the most-general concept names that are subsumed by the concept C, mentioned in the TBox.

A *coherence check* tests if all concept names mentioned in the TBox are consistent without computing the concept hierarchy. A concept C is consistent w.r.t. a TBox if there exists a model of C that is also a model of the TBox [7].

A service called *instance checking* tests if an individual i is an instance of a concept C. This is the case if $i^I \in C^I$ holds for all models of the TBox [7].

Direct Types are the most-specific concept names in a TBox that an individual is an instance of.

If all models of a TBox T and all models of an ABox A are also models of an ABox A' than we say that A' is *entailed* by T and A and we write $T \cup A \models A'$. This problem is called *ABox entailment* [7]. We show later on how this problem can be solved by a query answering service and in the next chapter we will argue why this problem corresponds to our main goal of finding a semantic matching for semantic descriptions.

2.2.2 Retrieval Inference Services

In our semantic matching task and also in other practical applications, a DL-reasoner needs to find individuals in an ABox that satisfy certain conditions. We discuss the basic retrieval inference service a conjunctive queries in the following.

The problem to find all individuals in an ABox that are instances of a given concept C or all individuals that are related via a certain role R is called *retrieval* inference problem. Sometimes one would like not only to retrieve individuals that are asserted to be an instance of a certain concept or a certain role but also to express complex relational structures. Therefore a more expressive query language such as conjunctive queries is required.

A query consists of a *head* and a *body*. The query head lists variables to which the retrieved individuals are bound. The query body consists of so-called *query atoms*. All variables listed in the head have to appear in at least one of the query atoms. A query has the form: $(X_1, \ldots, X_n)|atom_1, \ldots, atom_m$ [7].

A *query atom* may be a concept query atom ($C(X)$), a role query atom ($R(X, Y)$), a same-as query atom ($X = Y$)) or a so-called concrete domain query atom. Complex queries can be built from query atoms by using the following boolean operators: conjunction (indicated by commas), union ($\vee$) and query atom negation ($\neg$).

As an example we provide a complex query that asks for all running competitions and all athletes that are taking part in these competitions:

$$(X, Y)|RunningCompetition(X), Athlete(Y), hasParticipant(X, Y) \tag{2.1}$$

Queries of such a form are called conjunctive queries. In literature (e.g. [8],[9] and [10]) two different kinds of semantics are defined for conjunctive queries. In *standard* conjunctive queries, variables are bound to named domain objects which may be anonymous. In *grounded* conjunctive queries variables are only bound to domain objects that are named in the ontology / KB.

As already mentioned, ABox entailment can be solved by using query answering. A Boolean query is a query with no variables in the head. An ABox A' is entailed by a TBox T and an ABox A ($T \cup A \vDash A'$) if for all assertions α in A' it holds that the boolean query $\{()|\alpha\}$ returns *true* as an answer [7]. This is a very important fact for this work.

2.2.3 Non-standard Inference Services

Besides the standard inference services and retrieval inference services, there are some non-standard inference services that are implemented by DL-reasoners. We describe only *abduction* as it is important for this work. Please refer to the literature for more informations about other non-standard inference services [7].

Abductions tries to find a set of explanations Δ for a given set of assertion Γ such that Δ is consistent w.r.t. to the ABox A and the TBox T and satisfies:

1. $T \cup A \cup \Delta \vDash \Gamma$

2. If Δ' is an ABox satisfying $T \cup A \cup \Delta' \vDash \Gamma$, then $\Delta \vDash \Delta'$

3. $\Gamma \not\models \Delta$

In this work we only focus on reasoning services that are implemented by practical DL-reasoners like RacerPro. More details on reasoning services can be found in [7] [11] and [12]

2.2.4 New Racer Query Language (nRQL)

The DL-reasoner RacerPro supports two kind of query languages for grounded conjunctive queries, namely new Racer Query Language (nRQL) and Simple Protocol and RDF Query Language (SPARQL) [12][13]. In this work we focus on nRQL because it not only allows us to formulate complex queries but also enables powerful so-called *miniLisp* programs that are executed within the DL-reasoner (server-side execution).

The query in 2.1 can be formulated in nRQL as follows:

```
(retrieve (?x ?y)
          (and (?x RunningCompetition)
               (?y Athlete)
               (?x ?y hasParticipant)))
```

The first line indicates that the query requires an instance retrieval for the variable bindings ?x and ?y. In the following line, the keyword `and` *conjuncts* all query atoms. Beside `and`, nRQL supports also other keywords as Boolean operators, namely `or` for the *union* and `neg` for the query atom *negation*.

Keep in mind that query atom negation refers to the complement of the query atom. It implements the *Negation as Failure Semantics* (NAF). E.g. `(neg (?y Athlete))` returns all individuals for which it cannot be proven that they are an instance of *Athlete*. More details on the NAF can be found in [13] and [12].

Besides instance retrieval, nRQL has a simple expression language called 'MiniLisp'. MiniLisp is a Lisp dialect and enables the user to write simple, termination-safe 'programs' that are executed on the server side, e.g. by RacerPro.

MiniLisp supports the following features:

- user-defined output formats for query results (e.g., query results can also be written into a file)

- certain kinds of combined ABox/TBox queries
- efficient aggregation operators (e.g., sum, avg, analogous to the Structured Query Language(SQL) in database systems)
- so-called lambda expressions, which can denote (anonymous) functions

Lambda expressions are very useful when it comes to process results of RacerPro functions on server side. For instance, if we get a list of individuals as a result of a function call and we are not interested in the individuals themselves but on their direct types, this can be efficiently solved on the server side by using lambda expressions.

In the next two chapters we show how query answering and abduction can be exploited in order to match different semantic descriptions of a multimedia document.

3 Approach 1: Precision and Recall

As already mentioned, in the BOEMIE project the semantic descriptions of media objects and multimedia documents are stored in the form of ABoxes.

Henceforth we denote an ABox where the manually annotated semantic descriptions are stored by $\mathcal{M}$ and an ABox where the system generated semantic descriptions are stored is denoted by $\mathcal{S}$.

In this chapter we propose an approach which computes measurement values that reveal how many assertions from a manually annotated ABox $\mathcal{M}$ are also in the system generated ABox $\mathcal{S}$.

The problem of ABox entailment $\mathcal{T} \cup A \models A'$ (presented in Section 2.2.1) represents exactly our challenge to compare the manually annotated ABox $\mathcal{M}$ with the system generated ABox $\mathcal{S}$. We assume that both ABoxes share the same TBox $\mathcal{T}$. Therefore we formulate our problem to:

$$\mathcal{T} \cup \mathcal{S} \models \mathcal{M}$$

This means that, if the system generated ABox $\mathcal{S}$ entails the manually annotated ABox $\mathcal{M}$, all information about a media object that is stated by the annotators can also be found in $\mathcal{S}$ and thus the information in $\mathcal{S}$ is at least as specific as stated in $\mathcal{M}$.

In Section 2.2.2 we mentioned that ABox entailment can be solved by query retrieval. We exploit this fact to compute a measurement value that describes 'how much' information in $\mathcal{M}$ is also found in $\mathcal{S}$. This is shown after a short introduction to precision and recall.

3.1 Definition of Precision and Recall

In the field of information retrieval, precision and recall have become common standards to measure the performance of an information retrieval system (IRS).

Normally these values measure how good documents of a certain kind can be retrieved from a large corpus of documents. E.g. if we have a corpus of landscapes and would like to know how good the performance of retrieving beach landscapes is, we send a query to our IRS and count the number of landscapes that our IRS gives us back (the *retrieved* ones), the number of landscapes that are retrieved and actually show a beach (the correctly retrieved) and all landscapes that are in the corpus and show a beach (the relevant ones). Now we can calculate precision and recall using the counted numbers.

Definition 1 (Recall) *Recall is the fraction between the number of the elements in the intersection of the relevant with the retrieved elements and the number of relevant elements* $R \in [0, 1]$.

$$R = \frac{|relevant \cap retrieved|}{|relevant|} \tag{3.1}$$

Definition 2 (Precision) *Precision is the fraction between the number of the elements in the intersection of the relevant elements with the retrieved and the number of retrieved elements* $P \in [0, 1]$.

$$P = \frac{|relevant \cap retrieved|}{|retrieved|} \tag{3.2}$$

In our case we do not examine a corpus of documents, but pairs of ABoxes. However these ABoxes can be seen as sets of assertions. Thus we can formulate relevant and retrieved elements regarding our two ABoxes $\mathcal{M}$ and $\mathcal{S}$.

Similar to the information retrieval scenario, we define all answers that can be retrieved from $\mathcal{M}$ using a query q as the *relevant* elements. Therefore the number of relevant elements $|relevant|$ is equal to the cardinality of the answers retrieved from ABox $\mathcal{M}$ using query q (or a set of queries Q).

Respectively, the number of retrieved elements $|retrieved|$ is equal to the cardinality of the answers retrieved from ABox $\mathcal{S}$ using query q (or a set of queries Q).

And the number of elements in the intersection of precision and recall $|relevant \cap retrieved|$, is the cardinality of answers that can be retrieved from ABox $\mathcal{M}$ as well as from ABox $\mathcal{S}$ using query q (or a set of queries Q).

With these definitions we are able to calculate *precision* and *recall* according to Formulas 3.1 and 3.2.

In the next section, we propose a process that first extracts a set of relevant assertions from $\mathcal{M}$ and a set of assertions from $\mathcal{S}$ that could be retrieved and then transforms the relevant assertions into *grounded conjunctive queries*. These queries are then retrieved from $\mathcal{S}$. Finally precision and recall are computed using the number of positively answered queries, relevant assertions and retrieved assertions.

3.2 The Process

Our evaluation process has four steps:

1. Extract 'structures' from $\mathcal{M}$ and $\mathcal{S}$ (*structure scan*).

2. Transform 'structures' from $\mathcal{M}$ into grounded conjunctive queries (*query transformation*).

3. Query $\mathcal{S}$ using queries from 2(*structure retrieval*).

4. Compute precision and recall.

We call 'structure' a set of all assertions that follow a specific characteristic and can be retrieved from an ABox by a query q (or a set of queries Q). For example two individuals, their datatype properties, their datatype fillers and the role assertion between them is a 'structure'. The structures should be defined in such a way that each role assertion belong to only one structure.

The next step, before computing precision and recall, is to transform these assertions into grounded conjunctive queries. As mentioned in Section 1.2, we cannot rely on the individual names. Therefore the queries have to be anonymized, which means that individual names are replaced by variables. This leads to the problem that there may be more than one structures in $\mathcal{S}$ fitting to the query. More precisely: We are not always able to identify an individual from $\mathcal{M}$ as a unique individual in $\mathcal{S}$.

However, datatype filler and media related information that refer to the individuals can help to identify individuals in $\mathcal{S}$.

3.3 Datatype Properties and Media-related Information

In the BOEMIE semantics extraction process, as described in Section 1.1, analysis generates descriptions for objects of low abstraction level, which can be directly identified in the media object. We refer to these objects as *surface-level objects*. If analysis identifies a surface-level object, it gives a unique name to that object and adds a concept assertion to the analysis ABox. In the TBox, used by the BOEMIE semantics extraction process, all concepts that can be identified by the analysis are *subsumed* by a concept called Mild Level Concepts (MLC). We refer to individuals created by the analysis as *surface-level individuals*.

In the text modality, analysis identifies named entities (strings) and asserts them to the ABox by exploiting datatype properties. For example if a text contains the string 'Isinbayeva', analysis identifies it as a named entity and states a concept assertion $PersonName(p_1)$ and also a datatype property $hasValue(p_1, `Isinbayeva')$ in the ABox. The string 'Isinbayeva' is used as a datatype filler for the datatype property.

Analysis also asserts relations between named entities in the ABox. For example if a performance is identified in a text section that is related to the named entity 'Isinbayeva', analysis also states a concept assertion $Performance(perf_1)$ and a role assertion
$PersonName2Performance(p_1, perf_1)$ in the ABox.

Besides the datatype filler in the text modality, in every modality media related informations are created and asserted with reference to individuals. For example in text modality, the positions of tokens are stored, in image modality the positions of regions of the extracted objects are stored. etc.

We can use this additional information to identify MLC instances.

In this work we take only the datatype fillers into account, because we focus on the text modality where datatype fillers are available. All proposed principals apply also if media related information are included.

manually annotated ABox $\mathcal{M}$	**system generated ABox $\mathcal{S}$**
$pName_1$:$PersonName$	x_1:$PersonName$
$pName_2$:$PersonName$	x_2:$PersonName$
$pName_3$:$PersonName$	x_3:$PersonName$
($pName_1$, '$Jaroslav\ Rybakov$'):$hasValue$	(x_1, '$Jaroslav\ Rybakov$'):$hasValue$
($pName_2$, '$Oskari\ Fronensis$'):$hasValue$	(x_2, '$Oskari\ Fronensis$'):$hasValue$

Figure 3.1: GS-ABox and SI-ABox Individuals with their datatype properties

As an example consider Figure 3.1 that shows two ABoxes: a manually annotated ABox $\mathcal{M}$ and a system generated ABox $\mathcal{S}$. In both ABoxes there are three individuals of the type $PersonName$, two with datatype filler and one without. The individual $pName_1$ from $\mathcal{M}$ can easily be identified as corresponding to x_1 in $\mathcal{S}$, because both individuals have the same datatype filler, namely '$Jaroslav\ Rybakov$'. The same applies to $pName_2$ and x_2.

The third individual in $\mathcal{M}$ and $\mathcal{S}$ has no datatype filler. To identify the corresponding individual in $\mathcal{S}$ it is not sufficient to ask for an individual of the type $PersonName$ in the ABox $\mathcal{S}$. This would result in an answer with all three individuals. Instead we have to formulate a query where an individual is bound to a variable that is of the type $PersonName$ and has *no value* for the datatype property $hasValue$. As x_3 is the only individual in $\mathcal{S}$ without a datatype filler, we get x_3 as the matching individual for $pName_3$.

Things become more complicated when it comes to identify corresponding individuals that are created by the interpretation process, because they do not have datatype fillers. These individuals are hypothesized during interpretation process and given unique names.

However, all hypothesized individuals are instances of concepts that are all subsumed by the concept HLC, which stands for 'High Level Concept'. We refer to individuals created by the interpretation process as *hypothesized individuals*. Hypothesized individuals represent abstract objects that are composed of surface-level objects. During the interpretation process hypothesized individuals are created to explain the occurrence of surface-level individuals. Role assertions are also introduced between the new hypothesized individuals and the surface-level individuals. To identify the corresponding hypothesized individual we can integrate all the surface-level individuals and their datatype fillers, which the hypothesized individual is related to.

Therefore we define for the purpose of evaluation so-called 'levels of abstraction' that specify the characteristics of the structures that are scanned in $\mathcal{M}$ and $\mathcal{S}$.

3.4 Structure Scan

In this section, we present how the levels of abstraction are applied to ABoxes and how structures are extracted from the ABox $\mathcal{M}$ and $\mathcal{S}$ depending on the level of abstraction.

As we need more information about the individuals e.g. their datatype properties and role fillers, we extract structures form an ABox by forming a so-called 's-expression' in MiniLisp that 'scans' the ABox for semantic structure of a certain kind, instead of using a simple queries. We call this kind of s-expression 'structure scan expression'. This structure scan expression is executed on the server site of RacerPro and is more efficient than using multiple queries to retrieve the same information.

Depending on the structure scan expression, the retrieved structures can consist of concept assertions, role assertions, datatype properties and datatype filler or unions of these.

3.4.1 Levels of Abstraction

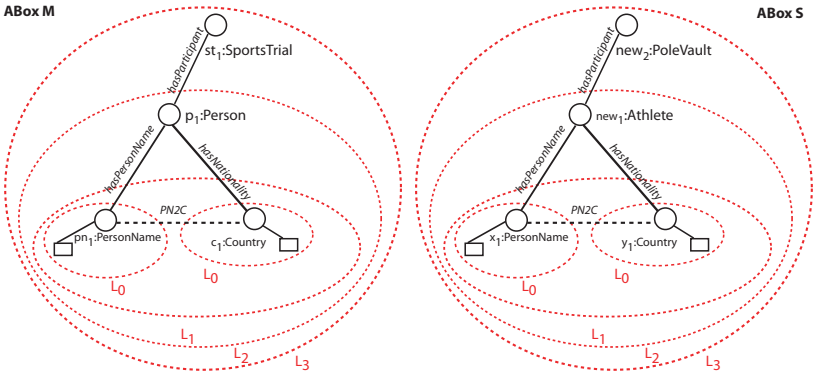

Figure 3.2: Levels of Abstraction in the manually annotated ABox $\mathcal{M}$ and the system generated ABox $\mathcal{S}$

Figure 3.2 shows the structures of abstraction levels L_0 to L_3 for the ABoxes $\mathcal{M}$ and $\mathcal{S}$. The small black circles in the graph indicate individuals and the black boxes indicate datatype fillers. The structures of each level are indicated by the red dashed lines. As one can see the structures of each level includes all concept assertions, role assertions and datatype fillers. Figure 3.2 also shows that the individual names in both ABoxes do not correspond to each other and the concepts asserted to individuals in $\mathcal{S}$ may be more specific than in $\mathcal{M}$.

In the following we present structure scan expressions for each level.

3.4.2 Abstraction Level 0

At the lowest level we examine how to match instances that have no children instances in both ABoxes $\mathcal{M}$ and $\mathcal{S}$. These instances result from the analysis process. We call these structures 'level 0'.

As mentioned, all individuals that are created by analysis are instances of the concept MLC. We apply the following structure scan expression to extract direct types and datatype properties with datatype fillers for all individuals in the ABox that are instances of MLC.

```
(evaluate  (progn
             (abox-consistent-p)
             (let* ((allMLCs
                     (retrieve-concept-instances MLC
                                                 (current-abox)))
                    (concepts
                     (maplist
                      (lambda (x)
                        (list
                         (most-specific-instantiators x
                                                      (current-abox))
                         (get-individual-datatype-fillers x)))
                      allMLCs)))
               concepts)))
```

Figure 3.3: Structure scan expression for level 0 structures

Figure 3.3 shows the structure scan expression, a MiniLisp program, for level 0 structures. In the first line the keyword `evaluate` instructs the DL-reasoner to execute the expressions as MiniLips programs. `progn` instructs that the following expressions are executed in sequence. In line two we instruct the DL-reasoner to do a consistency check. The next 12 lines executes a small program that gets all individuals that are instance of MLC and extracts their *direct types* and *datatype fillers*. The function `let*` assigns function results to a temporal variable. The function `retrieve-concept-instances` with the parameter `MLC` gathers all individuals that are instance of the concept MLC. With the *lambda expression* (see 2.2.4) we formulate a self defined anonymous function, that extracts the direct types (`most-specific-instantiators x`) of the individual assigned to the temporal variable `x` and the datatype fillers for the individual `x` (`get-individual-datatype-fillers x`).

3.4.3 Abstraction Level 1

In the next level, we scan for level 1 structures. Here we take pairs of surface-level instances that are related with each other. The structures look like $(j : MLC, i : MLC, (i, j) : r)$, where $j : MLC$,$i : MLC$ denotes that the two individuals are both instances of the concept MLC and $(i, j) : r$ denotes that they are related to each other.

The level 1 structure scan expression for this kind of structures is:

```
(evaluate
 (progn
   (abox-consistent-p)
   (remove nil
           (maplist
            (lambda (role)
              (let* ((class MLC)
                     (res (racer-retrieve-related-individuals role))
                     (mlcs (remove nil
                                   (maplist
                                    (lambda(pair)
                                      (when
                                          (and
                                           (individual-instance-p
                                            (first pair) class (current-abox))
                                           (individual-instance-p
                                            (second pair) class (current-abox)))
                                        (list
                                         (cons
                                          (most-specific-instantiators
                                           (first pair) (current-abox))
                                          (get-individual-datatype-fillers
                                           (first pair) (current-abox)))
                                         (cons
                                          (most-specific-instantiators
                                           (second pair) (current-abox))
                                          (get-individual-datatype-fillers
                                           (second pair) (current-abox))))))
                                    res))))
                (when mlcs (cons role mlcs))))
            (remove-if consp (all-roles))))))
```

Figure 3.4: Structure scan expression for level 1 structures

Figure 3.4 shows the structure scan expression for level 1 structures in nRQL syntax. To understand this expression we have to start at the last line, namely with the expression `(remove-if consp (all-roles))`. This expression creates a list of all roles that are not inverse roles. The result of this expression is then given to the first maplist function which iterates over all elements of the list and hands it over to the first lambda function `lambda (role)`. In the lambda function the function `racer-retrieve-related-individuals` results in a list of tuples of individuals that are related with the role assigned to the temporal variable `role`.

In a maplist expression and the second lambda function, each of these tuples is assigned to the variable `pair`. First we check if both individuals are instances of the concept MLC. This is done with the function `individual-instance-p`. If this test succeeds, we then extract the direct types and the datatype fillers of both individuals in the tuple. We give this information back together with the role name as a result for this level 1 structure scan expression `(cons role mlcs)`.

At Level 0 and level 1 we mainly evaluate the results from the semantic analysis process, but as they are the building blocks for abstract knowledge, they may be the cause for a missing or a misinterpreted result. An hypothesized individual is only constructed by the interpretation process, if there is a proper interpretation rule defined and all necessary surface-level individuals and roles have been generated by the analysis process. Therefore it is important to guarantee that all preconditions are given for the interpretation engine to produce abstract interpretations. Moreover to motivate learning, we would like to find the reason why an interpretation failed.

If instances or relations that are a presupposition for an aggregate concept are missing, then they can be definitely identified as the source for a missing interpretation. Otherwise the failure of interpretation is caused by missing rules or other causes.

3.4.4 Abstraction Level 2

Structures that represent aggregates and their relations that result from the interpretation process are considered starting from level 2. For level 2, we proceed in the same manner as for the lower levels but with the difference that we ask in the structure scan expression for hypothesized instances that are in a relation with surface-level instances.

In the TBoxes of the BOEMIE project the hypothesized individuals are all instances of the concept HLC, so to identify them we just have to ask for instances of the type HLC.

A structure scan expression for level 2 structures is stated in the following but its implementation details are not described here for the sake of brevity.

```
(evaluate (let* ((roles (all-roles))
                 (roles (remove-if consp roles))
                 (roles (remove-if (lambda (x)
                                     (role-used-as-datatype-property-p x
                                                                       (current-tbox))) roles)))
            (evaluate (cons 'progn
```

```
                          (maplist (lambda (x)
                                     `(role-has-parent (quote ,x)  'HLR (current-tbox)))
                                   roles)))))

(evaluate (let ((res (retrieve '(?x ?y) '(and (?x #HLC)
                           (?x ?y HLR)
                           (?y MLC)
                           (neg (project-to (?x)
                                            (and (?x ?z HLR)
                                                 (?z HLC))))))))
            (let ((inds (remove-duplicates (maplist (lambda (x)
                                                      (second (first x))) res))))
              (let ((res2 (maplist (lambda (ind)
                                     (list ind
                                           (maplist (lambda (x)
                                                      (second (second x)))
                                                    (remove-if (lambda (x)
                                                                 (not (equal
                                                                       (second (first x)) ind)))
                                                               res))))
                                   inds)))
                (maplist (lambda (x)
                           (list
                            (cons :HLC
                                  (most-specific-instantiators
                                       (first x) (current-abox)))
                            (maplist (lambda (mlc)
                                       (list
                                        (cons :relations
                                              (remove-if (lambda(role)
                                                           (role-equivalent-p role
                                                                'HLR (current-tbox)))
                                                         (retrieve-individual-filled-roles
                                                          (first x) mlc (current-abox))))
                                           (cons :MLC
                                                 (most-specific-instantiators
                                                      mlc  (current-abox)))
                                           (cons :D-Filler
                                                 (get-individual-datatype-fillers
                                                      mlc (current-abox)))))
                                     (second x))
                           )) res2)))))
```

Level 2 structure scan expression in nRQL syntax.

3.4.5 Abstraction Level ≥ 3

In the next level, namely level 3, we scan for structure for abstract hypothesized individuals that are composed of at least one structure of the level 2 and other structures of the level ≤ 2. This means we scan for hypothesized instances that are in relation to at least one hypothesized instance which is related only to surface-level instances.

Carrying this idea on for further steps we can define this algorithm recursively and say that a level n structure consists of at least one structure of the level n-1 (and other structures of the level ≤ n-1 for all n > 0).

3.5 Query Transformation

In this section we propose how to transform the results of structure scan expressions into grounded conjunctive queries. We described grounded conjunctive queries in general in Section 2.2.2.

As we are only interested if the structure we extracted from $\mathcal{M}$ is also available in $\mathcal{S}$, it is sufficient to formulate a Boolean query with query atoms that represents the information we extracted during the structure scan.

For example assume that the level 0 structure scan resulted in the following information:

```
(((PersonName))
        ((hasPersonNameValue
          ((d-literal "Slesarenko" (d-base-type #!xsd:string))))))
(((PersonName))
        ((hasPersonNameValue
          ((d-literal "Isinbayeva" (d-base-type #!xsd:string))))))
```

As we see there are two structures represented in this result, both are individuals of the type *PersonName* and have datatype properties of the name *hasPersonNameValue*. The first has the datatype filler `'Slesarenko'` and the second `'Isinbayeva'`.

To represent the first structure as a Boolean query we choose the variable `?x` for the individual name and formulate the query as following:

```
(retrieve () (and (?x PersonName)
                  (?x (string= hasPersonNameValue "Slesarenko"))))
```

The first query atom `(?x PersonName)` represents the concept assertion and the second query atom
`(?x (string= hasPersonNameValue 'Slesarenko'))` represents the datatype property with its role filler. Both query atoms are conjuncted with the a union indicated by the keyword `and`.

The same principals apply to the second structure but with the datatype filler 'Isinbayeva' instead.

If the structure includes more than one individual then we have to introduce a new variable name for each different individual.

For example let us assume the level 1 structure scan has in the following result:

```
((personNameToRanking
       ((((PersonName))
         (hasPersonNameValue
          ((d-literal "Marion Jones" (d-base-type #!xsd:string)))))
        (((Ranking))
         (hasRankingValue
          ((d-literal "5" (d-base-type #!xsd:string))))))
```

To represent this structure we have to introduce two variables: one for the individual of the type *PersonName* and one for the individual of the type *Ranking*. If we choose `?x1` as a variable for the first individual and `?x2` as a variable for the second one, our structure retrieval query would look as follows:

```
(retrieve () (and
              (?x1 PersonName)
              (?x1 (string= hasPersonNameValue "Marion Jones"))
              (?x2 Ranking)
              (?x2 (string= hasRankingValue "5"))
              (?x1 ?x2 personNameToRanking)))
```

The query atom `(?x1 ?x2 personNameToRanking)` states that the two individuals are related with each other by the role *personNameToRanking*.

The same principals apply also to structures of all further levels.

3.6 Structure Retrieval

After a structure is transformed into a Boolean query we can use the DL-reasoner to check if the structures we have extracted form $\mathcal{M}$ are also available in $\mathcal{S}$, by asking the DL-reasoner to query them in $\mathcal{S}$.

The DL-reasoner first tries to substitute the variable with individuals from the ABox and checks if all query atoms can be fulfilled for the variable substitution with respect to the TBox. If all query atoms can be verified as true then the result of the query is *true* otherwise *false*.

A positive result from the DL-reasoner means that the structure is in the intersection of relevant and retrieved. To calculate precision and recall we are interested in the cardinality of the intersection between the relevant and the retrieved structures. Therefore we simply count the number of all positively answered structure retrieved queries.

3.7 Calculation of Precision and Recall

To calculate precision and recall we need three important numbers:

1. The number of relevant structures in $\mathcal{M}$
2. The number of retrievable structures in $\mathcal{S}$
3. The cardinality of the intersection of relevant and retrieved structures.

To get the number of relevant structures we just count the structures of a specific kind after the structure scan. E.g in level 0 we count the number of structures that have the same direct type, in level 1 we count the number of structures that have the same relation etc.

To get the number of retrieved structures we apply the structure scan process to the ABox $\mathcal{S}$ and count similar structures by the same criteria as for the structures in $\mathcal{M}$.

Cardinality of the intersection of relevant and retrieved is the number of structures in $\mathcal{M}$ which structure scan query could also be positively answered in $\mathcal{S}$.

Sometimes the number of structures that are computed as correctly retrievable (intersection of relevant and retrieved) is greater than the number of the actual structures in $\mathcal{S}$. This is caused by the lack of a unique mapping between the individuals in $\mathcal{M}$ and the individuals in $\mathcal{S}$. This is especially the case if an individual cannot be distinguished by their direct types, datatype properties or multimedia information.

Without further restrictions, this is a violation of set theory regularities, namely that the number of elements in an intersection between two sets cannot be greater than the number of elements of one of the sets. This would also lead to errors in our calculations of precision and recall.

For example assume the following small ABoxes:

ABox $\mathcal{M}$	**ABox $\mathcal{S}$**
a_1:$AthleteName$	x_1:$PoleVaulterName$
a_2:$AthleteName$	
a_3:$AthleteName$	

With the corresponding TBox:

$$\begin{aligned} PersonName &\sqsubseteq MLC \\ AthleteName &\sqsubseteq PersonName \\ PoleVaulterName &\sqsubseteq AthleteName \end{aligned}$$

There are three $AthleteName$ instances in $\mathcal{M}$ but only one possible matching individual in the ABox $\mathcal{S}$. All instances in the ABox $\mathcal{M}$ and in the ABox $\mathcal{S}$ lack datatype properties. If we apply the process of structure scan, query transformation and structure retrieval on this example we get a positive matching of $a1, a2, a3$ with $x1$, because $PoleVaulterName \sqsubseteq AthleteName$ and therefore the structure retrieval queries result positively for all three individuals in $\mathcal{M}$.

With the definition the intersection we would get 3 for $|relevant \cap retrieved|$ and this would indicate to a recall of 1 at this level for this structure. Anyhow we know that there is only one instance in $\mathcal{S}$ which can match with at most one instance from $\mathcal{M}$.

Therefore we restrict our definition of the intersection between relevant and retrieved as follows:

$$|relevant \cap retrieved| \leq \min(|relevant|, |retrieved|) \tag{3.3}$$

Without this additional restriction the results of our example ABoxes would be:

concept name	$\|relevant\|$	$\|retrieved\|$	$\|relevant \cap retrieved\|$	P	R
$PersonName$	3	1	3	ERROR	1

The calculation of the *precision* value for our example would lead to an error because:

$$P = \frac{|relevant \cap retrieved|}{|retrieved|} = \frac{3}{1} = 3 \text{ ERROR!} \tag{3.4}$$

This is not possible, because $P \in [0, 1]$.

With the restriction we get a valid value for the precision in our example:

$$P = \frac{|relevant \cap retrieved|}{|retrieved|} S = \frac{1}{1} = 1 \tag{3.5}$$

With our restriction we get the following results:

conceptname	$\|relevant\|$	$\|retrieved\|$	$\|relevant \cap retrieved\|$	P	R
PersonName	3	1	1	1	0,33

The ambiguity in finding matching individuals may be distinguished when structures of a higher abstraction level are taken into account, because they may be distinguished by their relations or the individuals they are related to.

4 Approach 2: Evaluation with Abduction

In the last chapter, we showed how measurement values known in the field of information retrieval can be applied to ABox evaluation. In this chapter we present an approach where we use abduction to uncover the differences between two ABoxes. The principals of structure scan, query transformation and structure retrieval by queries are applied again here. But this time we submit our structure retrieval queries with the keyword 'retrieve-with-explanations' instead of just 'retrieve' to call the abduction service in the DL-reasoner.

Abduction is reasoning from effects to causes or from observations to explanations. Formally, abduction can be expressed as:

$$\mathcal{T} \cup \mathcal{A} \cup \Delta \models \Gamma \tag{4.1}$$

where $\mathcal{T}$ is the TBox and $\mathcal{A}$ is the ABox of the background knowledge and Γ the observations already made by analysis. The explanations (Δ) are to be computed by the abduction process. If description logics are used as the underlying knowledge representation formalism, Δ and Γ are ABoxes.

4.1 Omissions and Additions

In order to compare two ABoxes ($\mathcal{M}$,$\mathcal{S}$) for difference, consider this task as a logical entailment problem and adjust Formula 4.1 as follows:

$$\mathcal{T} \cup \mathcal{S} \cup \Delta_{\mathcal{S}-\mathcal{M}} \models \mathcal{M} \tag{4.2}$$

Now we can use abduction to derive explanations Δ for the given assertions in $\mathcal{M}$ such that Δ is consistent to w.r.t. $(\mathcal{T},\mathcal{S})$. In other words, what has to be added to $\mathcal{S}$ such that $\mathcal{M}$ is entailed in $\mathcal{S}$.

We call this set of assertions $\Delta_{\mathcal{S}-\mathcal{M}}$ *omissions*, because they are the assertions that are missing in $\mathcal{S}$ according to $\mathcal{M}$.

To discover the assertions that are in the system generated ABox $\mathcal{S}$ but are missing in $\mathcal{M}$, we apply the following formula:

$$\mathcal{T} \cup \mathcal{M} \cup \Delta_{\mathcal{M}-\mathcal{S}} \vDash \mathcal{S} \tag{4.3}$$

Since $\Delta_{\mathcal{M}-\mathcal{S}}$ is a set of assertions that exist in $\mathcal{S}$ but not in $\mathcal{M}$ we call this set *additions*.

4.2 Examples for Comparison of ABoxes

The steps of transforming the results of the structure scan and the retrieval process are omitted to keep track of the main idea. For more information on query transformation see 3.5.

Example 1: Low-Level Omissions

ABox $\mathcal{M}$	**ABox $\mathcal{S}$**
b_1:*Body*	x_1:*Body*
f_1:*Face*	y_1:*Face*
(b_1, f_1):*near*	
p_1:*Athlete*	
(p_1, b_1):*hasPart*	
(p_1, f_1):*hasPart*	

Figure 4.1: A defective image ABox $\mathcal{S}$ and the corresponding ABox $\mathcal{M}$

$$\begin{aligned}
Person &\sqsubseteq \exists hasPart.Body \sqcap \exists hasPart.Face \sqcap HLC \\
Athlete &\sqsubseteq Person \sqcap HLC \\
PoleVaulter &\sqsubseteq Athlete \\
SportsTrial &\sqsubseteq \exists_{\leq 1} hasParticipant.Athlete \sqcap HLC \\
Body &\sqsubseteq MLC \sqcap \neg Face \\
Face &\sqsubseteq MLC \\
PoleVault &\sqsubseteq SportsTrial \sqcap \forall hasParticipant.PoleVaulter \sqcap HLC
\end{aligned}$$

Figure 4.2: An simple TBox for ABoxes 4.1

As a example, consider the two ABoxes shown in figure 4.1 where the relation *near* between *Body* and *Face* is missing in the ABox $\mathcal{S}$ and therefore no individual of type *Person* could be interpreted as an explanation.

To detect this missing relation, we can evaluate the ABox through abduction as follows: First we check whether all level 0 (low-level) structures match. Following

the process steps of 'structure scan', 'query transformation' and 'structure retrieval' we prove that all low-level individuals are available in both ABoxes.

In the next level we evaluate whether all low-level relations between two low-level objects match. After structure scan and query transformation, we pose the following query to the DL-reasoner for answers form $\mathcal{S}$.

```
(retrieve
 ( )
 (and
  (?x Body)
  (?y  Face)
  (?x ?y near)))
```

We get the following result:

```
NIL
```

This means that the structure of an individual of the type *Body* that is in a relation *near* to an individual of the type *Face* cannot be retrieved from the ABox $\mathcal{S}$.

Now we use abduction to explain why this structure could not be retrieved. The abduction query look as follows:

```
(retrieve-with-explanation
 (?x ?y)
 (and
  (?x Body)
  (?y  Face)
  (?x ?y near))
 :only-best-p t
 )
```

Apart from the function name 'retrieve-with-explanation' and the abduction parameter ':only-best-p t', this query looks just like our original structure retrieval query. The parameter ':only-best-p t' state that we are only interested in the best solution, which plays only a role if more then one solution is computed by the DL-reasoner.

As a result for the abduction query we get the following:

```
1  (t
2         (((:tuple (?x #!:x1) (?y #!:y1))
3           (:new-inds)
4           (:hypothesized-assertions (related #!:x1 #!:y1 #!:near)))))
```

In the first line the keyword `t` indicates that an explanation could be computed. The second line shows variable bindings from the ABox $\mathcal{S}$ that match the query. In our example x_1 can be bound to the variable `?x` and y_1 to `?y`. Line 3 indicates new individuals that had to be hypothesized to answer the query. In our example there are no new individuals necessary, and thus nothing is listed here. From line 4 on, all assertions that have to be hypothesized are listed. The result shows that in the ABox $\mathcal{S}$ a relation *near* between x_1 and y_1 has to be hypothesized to answer the query with true.

This hypothesized assertion is missing in ABox $\mathcal{S}$ from Figure 4.1. Because the low-level relations, evaluated at level 1, should have been extracted in the analysis process, we can report this result as a feedback to the analysis module.

Example 2: High-level Omissions

ABox $\mathcal{M}$	**ABox $\mathcal{S}$**
b_1:*Body*	x_1:*Body*
f_1:*Face*	y_1:*Face*
(b_1, f_1):*near*	(x_1, y_1):*near*
p_1:*Athlete*	
(p_1, b_1):*hasPart*	
(p_1, f_1):*hasPart*	

Figure 4.3: ABox $\mathcal{S}$ is missing abstract objects.

As another example consider Figure 4.3 where both ABoxes match completely at the low-level but the ABox $\mathcal{S}$ is missing an abstract individual of the type *Athlete* and its relations. Tor this example there are no omissions and additions up to level 1. However at level 2 we will not be able to get an answer for the query that asks for a *Athlete* that has a *Body* which is *near* a *Face*.

```
(retrieve-with-explanation
 (?x ?y ?z)
 (and
  (?x Body)
  (?y Face)
  (?x ?y near)
  (?z Athlete)
  (?z ?x hasPart)
  (?z ?y hasPart))
 :only-best-p nil
 :show-score-p nil
 :final-consistency-checking-p t
 )
```

Figure 4.4: Abduction Query for the Level 2 structure for ABoxes 4.3

```
1  (t
2        (((:tuple (?x #!:x1) (?y #!:y1) (?z IND-73))
3          (:new-inds IND-73)
4          (:hypothesized-assertions
5           (related IND-73 #!:x1 #!:hasPart)
6           (instance IND-73 #!:Athlete)
7           (related IND-73 #!:y1 #!:hasPart)))))
```

Figure 4.5: Result of the Abduction Query 4.4

Figure 4.4 shows the abduction query and figure 4.5 the corresponding result.

Line 2 of the result shown in figure 4.5 displays the individuals x_1 and y_1 that were found as bindings for the *Body* and *Face* variables and also introduces a new individual $IND-73$, which is also listed in line 3 in the list of new individuals. This proves that an individual of type *Athlete* and its relations to a face and a body are missing in the ABox $\mathcal{S}$, shown in Figure 4.3.

In this example all low-level structures match but the evaluation on the high-level structure detected omissions. It is very likely that this kind of missing interpretation is caused by a missing or incorrect defined interpretation rule. Therefore this result is an important feedback for the interpretation module.

Example 3: Additions

ABox $\mathcal{M}$	**ABox $\mathcal{S}$**
b_1:*Body*	x_1:*Body*
f_1:*Face*	y_1:*Face*
$(b_1,\ f_1)$:*near*	$(x_1,\ y_1)$:*near*
p_1:*Athlete*	new_1:*PoleVaulter*
$(p_1,\ b_1)$:*hasPart*	$(new_1,\ x_1)$:*hasPart*
$(p_1,\ f_1)$:*hasPart*	$(new_1,\ y_1)$:*hasPart*
$(hj_1,\ p_1)$:*hasParticipant*	$(new_2,\ new_1)$:*hasParticipant*
hj_1:*HighJump*	new_2:*PoleVault*

Figure 4.6: Missinterpretation in the ABox $\mathcal{S}$

$$
\begin{aligned}
Person &\sqsubseteq \exists hasPart.Body \sqcap \exists hasPart.Face \sqcap HLC \\
Athlete &\sqsubseteq Person \sqcap HLC \\
PoleVaulter &\sqsubseteq Athlete \\
SportsTrial &\sqsubseteq \exists_{\leq 1} hasParticipant.Athlete \sqcap HLC \\
Body &\sqsubseteq MLC \sqcap \neg Face \\
Face &\sqsubseteq MLC \\
PoleVault &\sqsubseteq SportsTrial \sqcap \forall hasParticipant.PoleVaulter \sqcap HLC \\
HighJump &\sqsubseteq SportsTrial \sqcap \neg PoleVault \sqcap HLC
\end{aligned}
$$

Figure 4.7: An simple TBox for ABoxes in figure 4.6

Sometimes, the interpretation engine interprets an object completely differently than it is stated in the manually annotated ABox $\mathcal{M}$ by the human annotators. The reason for this may be an incorrectly designed interpretation rule or an incorrectly analyzed object in the analysis ABox. This means that either the analysis module or the interpretation engine added something to the system generated ABox compared to the manually annotated ABox.

Using the Abduction process we can detect these additions and report them as feedback either to the analysis module or the interpretation module.

Figure 4.6 shows an ABox $\mathcal{M}$ and an ABox $\mathcal{S}$, where in the ABox $\mathcal{S}$ a person is interpreted as an abstract composition of a body and a face, but then is wrongly assigned as a participant of a pole vault competition.

To detect these additions we have to first scan the structure of the ABox $\mathcal{S}$, then to transform the results into grounded conjunctive queries and finally try to retrieve them from the ABox $\mathcal{M}$. This will already fail at level 1 when we try to retrieve an individual of type $PoleVaulter$.

```
(t
       (((:tuple (?x #!:b1) (?y #!:f1) (?z #!:p1))
         (:new-inds)
         (:hypothesized-assertions (instance #!:p1 #!:PoleVaulter)))))
```

Figure 4.8: Result of the abduction query for a PoleVaulter in the ABox $\mathcal{M}$ shown in 4.6

Figure 4.8 shows the result of the abduction query. The results states that the concept assertion $PoleVaulter(p_1)$ has to be added to $\mathcal{M}$ in order to answer the query with true. From the view point of evaluation this assertion has to be seen as an addition of the ABox $\mathcal{S}$ compared to the ABox $\mathcal{M}$.

However when we retrieve the direct type of p_1 which is $Athlete$, we observe that $PoleVaulter$ is subsumed by $Athlete$ ($PoleVaulter \sqsubseteq Athlete$). This detected addition may be a correct specialization, according to the TBox shown in Figure 4.7. This kind of subsumption checking becomes only necessary, if manual annotated ABox $\mathcal{M}$ is more general than the system interpreted ABox.

At level 2 we try to explain why there is a structure of a pole vault in the ABox $\mathcal{S}$ that cannot be retrieved from the ABox $\mathcal{M}$ using the following query:

```
1  (retrieve-with-explanation
2   (?x ?y ?z ?w)
```

```
3   (and
4    (?x Body)
5    (?y Face)
6    (?x ?y near)
7    (?z PoleVaulter)
8    (?z ?x hasPart)
9    (?z ?y hasPart)
10   (?w PoleVault)
11   (?w ?z hasParticipant))
12  :only-best-p nil
13  :show-score-p nil
14  :final-consistency-checking-p t
15  )
```

For which we obtain the following result:

```
(NIL NIL :warning-only-inconsistent-explanations)
```

This result indicates that only inconsistent explanations can be found. To nevertheless retrieve explanations we have to switch off consistency checking in the query, changing the line 14 of the abductive query to ':final-consistency-checking-p nil'. Per default ':final-consistency-checking-p t' checks at the end of the computation of an explanation whether the explanation is consistent w.r.t. the ABox and the TBox or not.

Without consistency checking the result is:

```
(t
        (((:tuple (?x #!:b1) (?y #!:f1) (?z #!:p1) (?w #!:st1))
          (:new-inds)
          (:hypothesized-assertions
           (instance #!:p1 #!:PoleVaulter)
           (instance #!:hj1 #!:PoleVault)))))
```

Here the abduction process hypothesizes that the individual hj_1 is of type $PoleVault$. However, since as the correct direct type of the individual hj_1 is $HighJump$, which is disjoint with $PoleVault$, this is only an solution hj_1 is inconsistent. As we are only interested in the difference between two ABoxes we can ignore this fact. However this kind of information may be interesting if the differences of ABoxes are processed further e.g. for computing an additional measurement value depending on the differences of ABoxes.

5 Other Matching Approaches

In this chapter we present some other approaches which also to deal with the problem of matching two ABoxes either based on graph or string matching. After an introduction the the techniques used in that approaches, we discuss if they could also be applied to our problems.

5.1 Graph Matching

Definition 3 (Subgraph) *Let* $G = [V, E]$ *and* $G' = [V', E']$ *be two graphs.*

G' *is called a* subgraph *of* G *if* $V' \subseteq V$ *and* $E' \subseteq E$.

Definition 4 (Isomorphism) *Two graphs* $G = [V, E]$, $P = [V', E']$ *are called* isomorphic*, if there exists a bijection* $\varphi : V \to V'$ *with* $xy \in E \Leftrightarrow \varphi(x)\varphi(y) \in E'$ *for all* $x, y \in V$. *[14]*

Verbalized:
We call two graphs *isomorphic* if they contain the same number of vertices witch are connected in the same way.

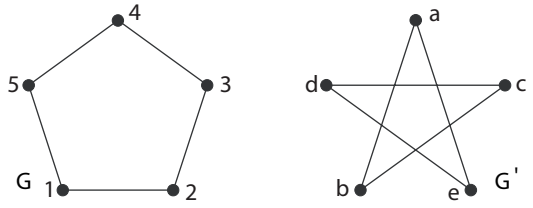

Figure 5.1: two graphs isomorphic to each other

Tim Berners-Lee identifies the task of finding the difference between two ontologies as a maximum common subgraph isomorphism problem[15].

5.1.1 Maximum Common Subgraph Isomorphism Problem

Given the two graphs G_1 and G_2 the maximum common subgraph isomorphism problem tries to find the largest induced subgraph of G_1 isomorphic to a subgraph of G_2.

5.1.2 RDF Diff

Tim Berners-Lee presents in [15] an approach, analogous to the text-diff that became popular in version control systems like CVS, to compare two RDF graphs with each other.

Triples of the RDF Graph are sorted by subject, and those that share a subject are grouped together. Then RDF graphs are serialized and 'pretty-printed' to text files. Now the two text files can be compared line by line with a standard text-diff tool.

Example

```
:Athlete     :hasGender :Female;
         :hasPersonName :PersonName[
             :hasPersonNameValue "Isinbayeva" ] .
```

Listing 5.1: "RDF Graph $\mathcal{M}$ after serialization and pretty-printing."

```
:PoleVaulter     :hasGender :Female;
         :hasPersonName :PersonName[
             :hasPersonNameValue "Isinbayeva" ] .
```

Listing 5.2: "RDF Graph $\mathcal{S}$ after serialization and pretty-printing."

5.1.3 Discussion

The *RDFdiff* technique proposed by Tim Berners-Lee lacks of an integration of the semantics and does not include information stated in the TBox. E.g. if the ABox $\mathcal{S}$ contains an individual of the type $PoleVaulter$ and a corresponding individual in the ABox $\mathcal{M}$ is of the type $Athlete$, this approaches would detect it as a mismatch. However a TBox $\mathcal{T}$, shared by both ABoxes might state that the concept $PoleVaulter$ is subsumed by the concept $Athlete$ ($PoleVaulter \sqsubseteq Athlete$).

As the BOEMIE semantics extraction process is designed to generate as specific informations about the media objects as possible this approach would detect many specializations as a mismatch.

5.2 Instance Matching

There are several instance matching tools that compute the similarity between two instances of two ontologies or a global value. This is mostly done by using string based similarity, angle cosine of the ontology features of the two individuals or a graph distance measurement of the concepts and roles of the instances [16]. In this chapter we present the functionalities of a popular ontology matching tool and discuss why computing just global or local similarity values is not sufficient for the evaluation of interpretation results.

5.2.1 HMatch 2.0

HMatch 2.0[1] is a tool for *mapping* on ontologies. Mainly developed for concept matching. HMatch's component HMatch(I) is nowadays able to match the ABoxes of two ontologies at the instance level.

In this section we present the operation method of HMatch(I) and discuss if it is suitable evaluation of system generated ABoxes.

The instance matching component of HMatch checks for two individuals the similarity of their direct types and their role fillers. The goal is to map to the same real object in two different ontologies. To realize this HMatch uses a similarity measure called *weighted dice coefficient*, that assigns some datatype fillers a factor representing their relevance. The assumption is that 'some properties are more important than others'[17].

The instance matching algorithm of HMatch 2.0 works in two steps:

1. An instance tree is constructed, that represents the ABox as a tree. The goal of this step is to represent the semantic dependencies between individuals through their roles.

[1] HMatch 2.0 is used in the BOEMIE Project for ontology mapping and population purposes.

2. The instance matching procedure: for each couple of of individuals a degree of similarity is computed.

   - The similarity between datatype values is generated and
   - it is propagated to higher level individuals, where they are combined together.

HMatch(I) uses a function *datatype role filler matching* that selects the most suitable matching technique for a role filler. Also HMatch(I) uses reasoning to find out all individuals in an ABox and to get all individuals of a given concept. The instance matching algorithm gets as input: 2 ABoxes, Mappings from a concept mapping process and a threshold. The threshold defines the minimal value of similarity that two individuals have to reach to be accepted as equal.

HMatch(I) uses a *top down* tree traversion to get down to the datatype fillers called 'depth first recursive visit'[17] and combines than in a *bottom up* process the similarity degrees of the datatype fillers.

The Procedure $MatchIndividuals(i_1, i_2)$ works as following:

1. Check if the concepts of the individuals are compatible.
2. Get all role of the individuals and check if the roles match.
3. Get role fillers of each role and execute $MatchIndividuals$ on each role filler. IF role filler is a datatype property THEN compute the similarity degree ELSE call $MatchIndividuals$ on the fillers.

To combine the similarity values of datatype properties HMatch 2.0 uses a so called 'weighted dice coefficient'. For each element of each ontology (e_i, e_j) a set of mappings $M_{e_i,e_j} =< e_i, e_j, R, V, S >$ is given. The mapping contains the semantic relation R which the two elements share (e.g. synonym, hypernyme, meronym, etc.), a confidence value V, which is the ratio between the number of occurrences of R in WordNet and the number of relationships occurrences retrieved between the two elements and S which denotes the similarity between the elements.

A set of similar elements S_{XY} is defined as the set of elements that have a similarity value in their mapping that is higher or equal then a threshold t.

$$S_{XY} \equiv \{e_i, e_j \mid e_i \in X, e_j \in Y, \exists \langle e_i, e_j, R, V, S \rangle \in M_{XY}, S > t\}$$

The weighted dice coefficient is calculated as follows:

$$WDC_{XY} = \frac{\sum_{e \epsilon S_{XZ}} W_e}{\sum_{k \epsilon X \cup Y} W_k} \tag{5.1}$$

The weights have to be defined manually by a domain expert. If no datatype property value is present than HMatch 2.0 makes an optimistic supposition and assumes the individuals to be equal if their direct types match according to the TBox matching of HMatch. This decision is taken, because it is assumed that the ontology is well defined and all existential restrictions must exist. An absence of other properties does not take in the similarity degree computation.

In a semantic clustering process the roles are classified according to the meaning they involved. It is distinguished between domain-independent features and domain-specific ones. A configuration ontology specifies which matcher is the right to be chosen for a datatype property class. During runtime the question of the one correct match is solved by a reasoning agent.

5.2.2 Discussion

Although HMatch 2.0 has the ability to compute a global matching value of two ontologies and can even give a similarity value for every pair of individuals, it is not sufficient to solve the challenge of finding all differences between two ABoxes. Moreover, the necessity of weighting the datatype properties can lead to the disadvantage that differences, in apparently not so relevant datatype fillers, are 'weighted out' and their individuals are taken as equivalent.

The advantage of HMatch 2.0 is when it comes to match two ABoxes that do not share the same TBox, but both TBoxes have linguistic similarities in their concept names. Then the linguistic mappings of the concepts can be used to identify equivalent individuals.

6 Results and Conclusion

In this chapter we present the results of our evaluation on a text document. We evaluate text results and conclude this work with a discussion of areas of future work.

6.1 Results

The following table presents the precision and recall values for the level 0 structure results (MLC instances):

concept name	$\|relevant\|$	$\|retrieved\|$	$\|relevant \cap retrieved\|$	P	R
$Date$	2	1	0	0.0	0.0
$Running100mName$	2	2	0	0.0	0.0
$SportsName$	12	10	10	1.0	0.83
$Male$	2	2	2	1.0	1.0
$Female$	2	2	2	1.0	1.0
$Ranking$	48	50	37	0.74	0.77
$HammerThrowName$	1	2	0	0.0	0.0
$SportsRoundName$	2	2	2	1.0	1.0
$HighJumpName$	2	2	0	0.0	0.0
$Performance$	5	5	5	1.0	1.0
$LongJumpName$	4	4	0	0.0	0.0
$PersonName$	62	63	61	0.97	0.99
$PoleVaultName$	2	2	0	0.0	0.0

The following table presents the precision and recall values for the level 1 structure results (relation between two MLC instances):

relation name	$\|relevant\|$	$\|retrieved\|$	$\|relevant \cap retrieved\|$	P	R
$performanceToRanking$	5	1	1	1.0	0.2
$sportsRoundNameToDate$	1	0	0	-	0.0
$personNameToRanking$	11	27	0	0.0	0.0
$personNameToGender$	15	64	4	0.06	0.27

The following table shows the precision and recall values for the level 2 structure results (HLC instances that are only in relation with MLC instances):

HLC concept name	$\|relevant\|$	$\|retrieved\|$	$\|relevant \cap retrieved\|$	P	R
Athlete	12	59	4	0.07	0.33

As one can observe in the result tables the values of precision and recall are low for level 1 and level 2 structures. Low values at a certain level influence the values of higher levels negatively if the structures of the higher level contain lower level structures as parts.

The result tables we present here were generated with a program that was developed in the context of this thesis which can compute precision and recall for collections of arbitrary length. A collection contains pairs of corresponding manually annotated and system generated ABoxes. The program can be adapted to any kind of ABoxes by adjusting the structure scan expressions (MiniLisp programs) to the specifications of the application context.

Due to time constrains we only computed precision and recall for one pair of ABoxes. However, using the program developed as part of this thesis the measurement values can be computed for a larger corpus of documents once all required data is available.

6.2 Conclusion

After an introduction into the BOEMIE interpretation process and presentation of the objectives of evaluating two ABoxes we proposed a new approach to match two ABoxes on the semantic level. To this end, we exploited grounded conjunctive queries and reasoning services offered by a DL-reasoner. We demonstrated how to compute the performance values like precision and recall and presented the results for a text interpretation ABox.

Although the numbers for precision and recall itself may dissatisfy, this work succeeded to provide a method to evaluate two ABoxes $\mathcal{M}$, $\mathcal{S}$ with respect to a TBox $\mathcal{T}$. The ability to evaluate system generated ABoxes makes it possible to measure the effects of changes in the background knowledge. For example let $\mathcal{S}_1$ be a system generated ABox that was generated before a change in the background knowledge

and $\mathcal{S}_2$ be a system generated ABox that is generated after a change in the background knowledge. We can see the effects of the change by computing the precision and recall values using $\mathcal{S}_1$ and $\mathcal{M}$ and using $\mathcal{S}_2$ and $\mathcal{M}$. This enables us in deciding whether $\mathcal{S}_1$ or $\mathcal{S}_2$ corresponds to more successful interpretation results.

Dividing the evaluation into levels of abstraction makes it possible to evaluate each step of the semantics extraction process separately. In this work, we applied the evaluation for ABoxes to semantic descriptions of media objects of the text modality. This approach could also be applied to ABoxes of other modalities if corresponding manually annotated ABoxes exist.

In a second approach, we showed that using grounded conjunctive queries and an abductive retrieval process, we can compute the semantic differences between two ABoxes. These differences can be presented to the developers of the analysis and interpretation modules as a feedback. The developers themselves can use the feedback to improve interpretation rules or adjust parameters of the employed analysis algorithms.

To examine the effects of changes in the background knowledge, rules or analysis parameters, it is also possible to evaluate the differences between two system generated ABoxes $\mathcal{S}_1$, $\mathcal{S}_2$ with the same approach as the one applied for the ABoxes $\mathcal{M}$ and $\mathcal{S}$. Further the resulted omissions and additions can be used to devise a measurement value for the difference of two ABoxes. This measurement value could than be integrated into the evaluation report as additional information.

As a result of the discussions during this work a function called *'compute-abox-difference'* is implemented in the latest version of RacerPro[1]. Although the algorithm is not scaling very well yet for large ABoxes, it works fine for smaller ones.

6.3 Future Work

6.3.1 Matching with different TBoxes

Using a TBox matching tool, like HMatch 2.0, the presented two approaches of semantic evaluation could be extended to evaluate two ABoxes that do not share the same TBox, but have linguistic similarities in their concept names.

[1] http://www.racer-systems.com/

6.3.2 Learning Rules with Evaluation Results

The omissions and the additions computed during the evaluation with abduction could be used to learn new interpretation rules and to optimize existing ones. An automatic rule learning process, which learns rules from the differences between two interpretation ABoxes $\mathcal{S}_1$ and $\mathcal{S}_2$ would enable a bootstrapping on the interpretation process. For example we begin with a analysis ABox $\mathcal{A}_1$ and interpret it by the interpretation process with the manually designed rule set $\mathcal{R}_M$. The result of this interpretation is the system generated ABox $\mathcal{S}_1$. Computing the differences between $\mathcal{S}_1$ and a manually annotated ABox $\mathcal{M}$, the rule learning process suggests a new rule set $\mathcal{R}_{A1}$. Now we can interpret the analysis ABox again, this time with the rule set $\mathcal{R}_{A1}$. The result is another system generated ABox $\mathcal{S}_2$. Comparing the two system generated ABoxes $\mathcal{S}_1$ and $\mathcal{S}_2$, we can compute the differences between them which can again be used to suggest an other rule set $\mathcal{R}_{A2}$ and so on.

Interpretation rules represent additional constrains to concepts in the TBox. During the interpretation process the rules are transformed into abduction queries and consistent results are then added into the interpretation ABox. [5] Nowadays these rules are designed manually by a human expert. The results of structure retrieval query results could be used for learning the rules automatically.

Bibliography

[1] M. K. Bergman, "The deep web: Surfacing hidden value." http://www.brightplanet.com/images/stories/pdf/deepwebwhitepaper.pdf, September 24, 2001.

[2] A. Gulli and A. Signorini, "The indexable web is more than 11.5 billion pages." http://www.cs.uiowa.edu/ãsignori/web-size/size-indexable-web.pdf, 2005, May.

[3] M. de Kunder, "The size of the world wide web." http://www.worldwidewebsize.com/, October 29, 2008.

[4] Y. Jing and S. Baluja, "Pagerank for product image search," *WWW 2008 / Refereed Track: Rich Media*, April 21-25, 2008. Beijing, China.

[5] S. Castano, A. F. Sofia Espinosa, V. Karkaletsis, A. Kaya, R. Möller, S. Montanelli, G. Petasis, and M. Wessel, "Multimedia interpretation for dynamic ontology evolution," *Journal of Logic and Computation*, 2008.

[6] F. Baader, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridges University Press, 2003.

[7] S. Castano, K. Dalakleidi, S. Dasiopoulou, S. Espinosa, A. Ferrara, G. N. Hess, V. Karkaletsis, A. Kaya, S. Melzer, R. Möller, S. Montanelli, and G. Petasis, "Methodology and architecture for multimedia ontology evolution," *BOEMIE deliverable, www.boemie.org*, 2006.

[8] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler, "Conjunctive query answering for the description logic SHIQ," *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07. AAAI Press*, 2007.

[9] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies, *How to decide query containment under constraints using a description logic*. In Proc. of the 7th Int. Conf.

on Logic for Programming and Automated Reasoning LPAR 2000, volume 1955 of Lecture Notes in Computer Science, Springer, 2000.

[10] M. Wessel and R. Möller, "A flexible DL-based architecture for deductive information systems," *in G.Sutcliffe, R. Schmidt, and S.Schultz, editors, Proc. IJCAR-06 Workshop on Empirically Succesful Computerized Reasoning (ESCoR), pp 92-111*, 2006.

[11] S. Espinosa1, V. Haarslev, A. Kaplunova, A. Kaya, S. Melzer, R. Möller, and M. Wessel, "D4.2: Reasoning engine version 1 and state of the art in reasoning techniques," *www.boemie.org*, Februrary 16, 2007.

[12] S. Espinosa, A. Kaya, S. Melzer, R. Möller, T. Näth, and M. Wessel, "D4.5: Reasoning engine - version 2," *BOEMIE Deliverable, www.boemie.org*, 2007.

[13] Racer Systems GmbH & Co. KG, "RacerPro User's Guide Version 1.9.2," *http://www.racer-systems.com*, October 18,2007.

[14] R. Diestel, *Graph Theory.* Springer-Verlag Heidelberg, 2005.

[15] Tim Berners-Lee and Dan Connolly, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), "Delta: an ontology for the distribution of differences between rdf graphs." http://www.w3.org/DesignIssues/Diff, Created: 2001, current: Revision: 1.114 of Date: 2006/05/12 23:07:43.

[16] C. Wang, J. Lu, and G. Zhang, "Integration of ontology data through learning instance matching," *IEEE/WIC/ACM International Conference*, NSW 2007, Australia.

[17] S. Bruno, S. Castano, A. Ferrara, D. Lorusso, G. Messa, and S. Montanelli, "Ontology coordination tools," *BOENIE Deliverable D4.7*, 2007.

[18] T. Berners-Lee, "Plenary at WWW Geneva 94." http://www.w3.org/Talks/WWW94Tim/, 1994.

[19] S. Petridis, N. Tsapatsoulis, D. Kosmopoulos, Y. Pratikakis, V. Gatos, S. Perantonis, G. Petasis, P. Fragou, V. Karkaletsis, K. Biatov, C. Seibert, S. Espinosa, S. Melzer, A. Kaya, and R. Möller, "D2.1 methodology for semantics extraction from multimedia content," *www.boemie.org*, December 21, 2006.

[20] G. Chartrand, *Introductory Graph Theory.* New York: Dover, 1985.

[21] E. M. Luks, "Isomorphism of graphs of bounded valence can be tested in polynomial time," *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, 1980.

List of Figures