Description Logic Query Answering with Relational Databases

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Norbert Kottmann

2006

Leiter der Arbeit:

Prof. Dr. Gerhard Jäger,
Institut für Informatik und angewandte Mathematik

Abstract

A knowledge base system O using description logics should be able to find an answer to the retrieval problem in reasonable time, returning a sound and complete result. That means finding all individuals a of a concept C where O entails that a belongs to C. We present a description logic based solution for a knowledge base where the information and its consequences, computed by a completion algorithm, are stored in a relational database. Our system issues simple database lookups to answer description logic queries. We are able to show that these queries can be answered in constant amount of time even for large data sets and complex concepts.

Acknowledgements

First, I would like to thank Prof. Dr. Gerhard Jäger, head of the Research Group for Theoretical Computer Science and Logic, for giving me the opportunity to work in his group.
Special thanks go to my supervisor Dr. Thomas Studer for his great support and motivation.
I would also like to thank Yuanbo Guo from Lehigh University, Bethlehem (PA), USA for his support on LUBM and Dr. Dieter Probst for his help and patience.
Furthermore, many thanks to my best friends Daria Spescha and Marco Robertini for their encouragement during my studies and for proofreading this thesis.
Finally, I would like to thank all members of the TIL group, it was a pleasure to work with them.

Norbert Kottmann
July 2006

Contents

1 Introduction

A knowledge base system has to fulfil different needs. On the one hand, it should be able to store and update existing information, to provide automated deductive reasoning and to return sound and complete results from queries. On the other hand, the query answering must be fast. This master thesis describes an implementation of a knowledge base system which uses description logics for logical inference and a relational database system for persistent storage. Hereby, the focus should lie on fast, sound and complete query processing because this is the most extensive feature of knowledge bases. The thesis builds on the paper "Relational Representation of $\mathcal{ALN}$ Knowledge Bases"[12] by Thomas Studer which provides the main concepts for the implementation.

As mentioned above, description logics and a relational database management system (RDBMS) will be used in our knowledge base application for information representation. This combination leads to a problem: A database works as a *closed world*, where absence of information is interpreted as negative information whereas description logics are characterised by an *open world* assumption, where absence of information means lack of knowledge. Therefore, we can not simply store the information in the database and issue queries on it as this may lead to incomplete query results, for example for the query "find me all individuals a which are of type C". In description logics, this task is known as the *retrieval problem* [2] and it is formulated as: "find all individuals a (of a given knowledge base O and a concept C) such that we can logically infer from O that a belongs to C". Figure 1.1 shows two different approaches to solve the retrieval problem: completion during loading or reasoning at query time.

Most of the existing knowledge base systems use a reasoner such as RacerPro[1] or FaCT[2] to answer the queries. Some of them solve the retrieval problem by a trivial generate and test method at query time, which tests for each individual in O whether it is an instance of the concept C or not.

Thomas Studer presents in his paper a completion based approach. The proposed solu-

[1] `http://www.racer-systems.com/`
[2] `http://www.cs.man.ac.uk/~horrocks/FaCT/`

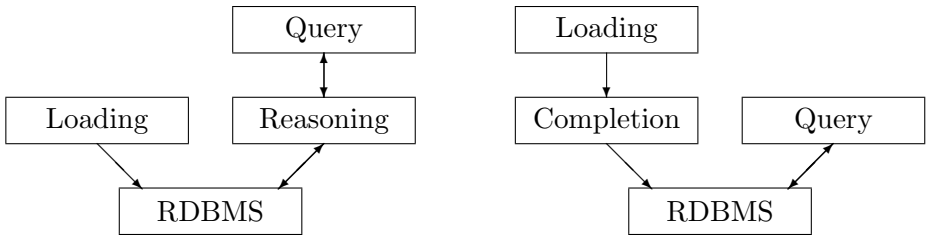

Figure 1.1: Load and query procedure of two different approaches. On the left hand side, the reasoning is done at query time and on the right hand side the completion happens at load time.

tion is formulated for the description logic language $\mathcal{ALN}$ where a completion algorithm computes the missing consequences before the information is stored in the database. The main advantage of this completion approach is, that even complex queries can be answered fast by a simple database query because the reasoning has already been done. Moreover, database optimisations can even improve the performance. One obvious disadvantage of such a system is the bad load time and storage complexity. Of course, a performance evaluation is needed in order to verify these assumptions.

1.1 Structure of the Document

Chapter 2 of this thesis introduces all necessary theoretical background for understanding the completion approach. A specific implementation is then presented in Chapter 3 which is based on the theoretical concepts. Furthermore, the performance evaluation of the implemented knowledge base application and a comparison to another system is given in Chapter 4. Finally the conclusions drawn from the results as well as thoughts about future work are summarised in Chapter 5.

2 Theory

As mentioned in the introduction, our aim is to build a knowledge base using a relational database. A knowledge base "is a special kind of database for knowledge management [...]. It provides the means for the computerized collection, organization, and retrieval of knowledge. [...] A knowledge base may use an ontology to specify its structure (entity types and relationships) and its classification scheme. An ontology, together with a set of instances of its classes constitutes a knowledge base."[5] "In general, an ontology describes formally a domain of discourse. Typically, an ontology consists of a finite list of terms and the relationships between these terms"[1].

First, to build a knowledge base, we need an ontology language to formalise the domain. The Web Ontology Language (OWL) is ideal for this purpose, because it is machine processable. Thus, a short description of OWL will be given in Section 2.1.

Since a description logic based completion algorithm will compute the knowledge base consequences, we have to define a description logic language for knowledge representation and completion. This is done in Section 2.2.

Finally, the completed knowledge base has to be stored in a relational database. This database conversion and the conversion between the OWL- and description logic representation layer as well as the completion rules will be presented in Section 2.3.

An ontology about the domain "species"will be used as an example throughout the chapter.

2.1 Web Ontology Language (OWL)

The Web Ontology Language (OWL) was developed by the W3C Web Ontology Working Group[1] in the context of the Semantic Web [1]. They describe OWL as a language which is "intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented

[1] `http://www.w3.org/2001/sw/WebOnt/`

to humans"[9]. OWL allows us to formalise information of a domain by creating classes, individuals and properties about the classes and individuals. OWL builds on RDF[2] and its syntax is based on RDF's XML syntax. A formal semantic is a prerequisite for reasoning tasks and was a requirement of OWL [8]. OWL allows references to other ontologies since every resource in an ontology is defined by an ID (`rdf:ID`, `rdf:about`) and a namespace (`xmlns:prefix`) to avoid name clashes [4] (`rdf` in `rdf:about` is the namespace prefix for the RDF ontology).

A class in OWL (`owl:Class`) describes a set of individuals which share some properties. For example *lion* and *zebra* are both instances (`rdf:type`) of the class *Animal*. OWL contains two predefined classes, the universal class (`owl:Thing`) containing all elements and the empty class (`owl:Nothing`). Subclass (`owl:subClassOf`), equivalence (`owl:equivalentClass`), intersection (`owl:intersectionOf`) and union (`owl:unionOf`) are axioms, which relate classes to others. For example *Plant* is a *subClassOf Creature*. A class can also be defined as a restriction on a property (`owl:Restriction`). This particular restriction could be a cardinality (`owl:cardinality`) or a type (`owl:someValuesFrom`, `owl:allValuesFrom`) restriction. Such a class is called "anonymous class"because it has no name. A value restriction for example could be: The class *Carnivore* is equivalent to the anonymous class of those individuals, which *eat* only individuals of the class *Animal*.

An object property (`owl:ObjectProperty`) can be used to state relationships between individuals whereas a datatype property (`owl:DatatypeProperty`) links individuals to data values. A *lion* can *eat* (object property) a *zebra*. Furthermore, a *lion* is related to the number *150* by the datatype property *averageWeight*. Properties can also have a hierarchy (`owl:subPropertyOf`), be equivalent (`owl:equivalentProperty`) or inverse (`owl:inverseOf`). For example the inverse of the object property *eats* may be defined as *eatenBy*.

Annotations, such as `rdfs:comment`, can be used to add additional information to classes, individuals or properties. It is also possible to define new annotations.

There are many other OWL and RDF statements defined by W3C. For a full list of available axioms, see the "OWL Web Ontology Language Reference"[3].

In order to meet different requirements concerning expressiveness and computational complexity in reasoning, OWL was divided into three sublanguages:

OWL Full: OWL Full contains the entire OWL language which uses all axioms. In OWL Full, one can even change the meaning of the predefined OWL or RDF primitives.

[2] `http://www.w3.org/RDF/`

Furthermore, a class can be treated as an individual. Therefore, OWL Full has the maximum expressivness but is undecidable.

OWL DL: As a sublanguage of OWL Full, OWL DL (DL for Description Logic) is designed as a trade-off between computational efficiency and expressiveness, which corresponds to the one of description logics. Unlike OWL Full, classes, properties, individuals and datatypes are disjoint.

OWL Lite: OWL Lite has minimal expressiveness and is a sublanguage of OWL DL. Compared to OWL DL, arbitrary cardinality constrains are for example not allowed.

As we will see in Section 2.3.1, a subset of OWL Lite is sufficient for building our knowledge base.

2.1.1 Example

The OWL code in Listing 2.1 shows a small ontology about species. Lines 10 to 18 set the RDF namespaces and their prefixes. The class hierarchy between *Animal* and *Creature* is given in lines 21 to 26 where *Animal* is defined as a subclass of *Creature*. Lines 27 to 51 formalise that any *Omnivore eats* at least one individual of the class *Plant* and the class *Animal*. This is defined by an equivalence to an anonymous class which is a conjunction of two restrictions. There is also a comment annotation in line 50. The object property (lines 52 to 61) introduces a role *eats* with its domain and range being individuals from the class *Creature*.

```
1  <?xml version="1.0"?>
   <!DOCTYPE owl [
3  <!ENTITY creatures "http://www.iam.unibe.ch/~kottmann/creatures#">
   <!ENTITY dc "http://purl.org/dc/elements/1.1/">
5  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
7  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
9  ]>
   <rdf:RDF
11   xmlns:creatures="http://www.iam.unibe.ch/~kottmann/creatures#"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
13   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
15   xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
17   xml:base="http://www.iam.unibe.ch/~kottmann/creatures#"
   >
```

```
<owl:Ontology rdf:about="">
</owl:Ontology>
<owl:Class rdf:about="#Animal">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Creature">
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Omnivore">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#eats" />
          <owl:someValuesFrom>
            <owl:Class rdf:about="#Plant">
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Class rdf:about="#Animal">
        </owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#eats" />
          <owl:someValuesFrom>
            <owl:Class rdf:about="#Animal">
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:comment>Animal that eats plants and meat.</rdfs:comment>
</owl:Class>
<owl:ObjectProperty rdf:about="#eats">
  <rdfs:domain>
    <owl:Class rdf:about="#Creature">
    </owl:Class>
  </rdfs:domain>
  <rdfs:range>
    <owl:Class rdf:about="#Creature">
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
</rdf:RDF>
```

Listing 2.1: Example of an OWL ontology about species.

2.2 Description Logic Language (DL)

As basis for our knowledge base, we need to define the syntax and semantics of the underlying description logic language. The language differs from $\mathcal{ALN}$ employed in the paper "Relational Representation of $\mathcal{ALN}$ Knowledge Bases"[12], because the final benchmark test uses the "Univ-Bench"ontology (Section 4.1.1), which has different requirements of the description logic language. For a detailed introduction to description logics, read "The Description Logic Handbook"[2].

2.2.1 Syntax

The syntax of our description logic language is given by the following rules, where A denotes an *atomic concept*, C, D are used for *concept descriptions*, S, T represent *atomic roles* and R stands for a *role*:

$C, D \rightarrow$	$A \mid$	(atomic concept)
	$\top \mid$	(top concept)
	$C \sqcap D \mid$	(conjunction)
	$\exists R.C \mid$	(full existential quantification)
	$\forall R.C \mid$	(value restriction)
$R \rightarrow$	$S \mid$	(atomic role)
	$S^{-} \mid$	(inverse role)

Additionally, we consider a subset R^{+} of roles. The elements of R^{+} are called *transitive roles*.

Note that there is neither an atomic negation nor a bottom concept. We use full existential quantification where arbitrary concepts are allowed to occur in the scope of the existential quantifier. Furthermore, a role can either be atomic, inverse or transitive.

2.2.2 Semantics

An interpretation I consists of a pair $(\Delta^I, \cdot^I)$. Δ^I is called the domain of interpretation and is a non-empty set of individuals and $\cdot^I$ is an interpretation function assigning to each atomic concept A a set $A^I \subset \Delta^I$ and to each atomic role S a binary relation $S^I \subset \Delta^I \times \Delta^I$ where a role $R \in R^{+}$ must be mapped to a transitive role R^I. We extend the interpretation function to concept descriptions and roles by the following inductive definition:

$$
\begin{aligned}
\top^I &:= \Delta^I \\
(C \sqcap D)^I &:= C^I \cap D^I \\
(\forall R.C)^I &:= \{a \in \Delta^I : \forall b((a,b) \in R^I \rightarrow b \in C^I)\} \\
(\exists R.C)^I &:= \{a \in \Delta^I : \exists b((a,b) \in R^I \wedge b \in C^I)\} \\
(S^-)^I &:= \{(b,a) \in \Delta^I \times \Delta^I : (a,b) \in S^I\}
\end{aligned}
$$

Note that the interpretation of the transitive role is not equal to the transitive closure. As we will see later, we only need to mark a role as transitive for the completion rules (Section 2.3.3).

2.2.3 Knowledge Base O

We are now able to define a description logic knowledge base O. The knowledge base O consists of a *terminology* $O_{\mathcal{T}}$ called *TBox* and a *world description* $O_{\mathcal{A}}$ called *ABox*.

Terminology $O_{\mathcal{T}}$

The TBox contains *concept definitions* and *inclusions*, whereas inclusions can be concept inclusions or role inclusions:

- *Concept Definitions:* $A := C$
- *Inclusions:* $C \sqsubseteq D$, $S \sqsubseteq T$

Before we can introduce the terms acyclic or cyclic, we need to divide the atomic concepts occuring in $O_{\mathcal{T}}$ into two sets:

- *name symbols* $N_{\mathcal{T}}$*:* atomic concepts that occur on the left hand side of a concept definition.
- *base symbols* $B_{\mathcal{T}}$*:* atomic concepts that occur only on the right hand side of the concept definitions.

For example if $O_{\mathcal{T}} := \{A := B \sqcap C, B := C \sqcap D\}$ then $N_{\mathcal{T}} := \{A, B\}$ and $B_{\mathcal{T}} := \{C, D\}$.

Let A and B be atomic concepts. A *depends* on B if B appears in the concept definition of A. A terminology $O_{\mathcal{T}}$ is called *acyclic* if there is no cycle in this dependency relation. Otherwise the terminology is called *cyclic*. For example $O_{\mathcal{T}} := \{A := B \sqcap C, B := A \sqcap D\}$ is cyclic whereas $O_{\mathcal{T}} := \{A := B \sqcap C, B := C \sqcap D\}$ is acyclic.

As we do not restrict concept inclusions to atomic concepts, it is necessary that there is no cycle in the inclusion hierarchy (there is no chain $L_0, L_1, L_2, ..., L_n$ such that L_0 is the same concept as L_n and all statements $L_i \sqsubseteq L_{i+1}$ are included in the TBox. The same condition must hold for role inclusions.

The semantics of the TBox is given as follows: an interpretation I satisfies $O_{\mathcal{T}}$ if:

- for every concept definition $A := C$ we have $A^I = C^I$
- for every concept inclusion $C \sqsubseteq D$ we have $C^I \subseteq D^I$
- and for every role inclusion $S \sqsubseteq T$ we have $S^I \subseteq T^I$

World Description $O_{\mathcal{A}}$

The world description introduces a set of *individuals* and assertions about them. The set of individual constants are denoted by $a, b, c, ...$ The ABox contains the following assertions:

- *Concept Assertions:* $C(a)$
- *Role Assertions:* $R(b, c)$

A *concept assertion* states that an individual a belongs to the interpretation of C, whereas a *role assertion* means that b is related to c by the role R.

For formulating the semantics of the ABox, the interpretation function $\cdot^I$ is extended by mapping each individual constant a to an element a^I of Δ^I. The interpretation function should respect the *unique name assumption*, that distinct individual constants denote distinct elements in Δ^I. Therefore, if a and b are different individuals, then they have a different interpretation $a^I \neq b^I$. Altogether, the semantics of the ABox assertions is given as follows: an interpretation I satisfies $O_{\mathcal{A}}$ if:

- for every concept assertion $C(a)$ we have $a^I \in C^I$
- for every role assertion $R(b, c)$ we have $(b^I, c^I) \in R^I$

2.3 Load and Query Process

After defining the syntax (Section 2.2.1), the semantics (Section 2.2.2) of the description logic language, the corresponding knowledge base (Section 2.2.3) and the OWL ontology language (Section 2.1), we can now describe the whole loading process (Figure 2.1).

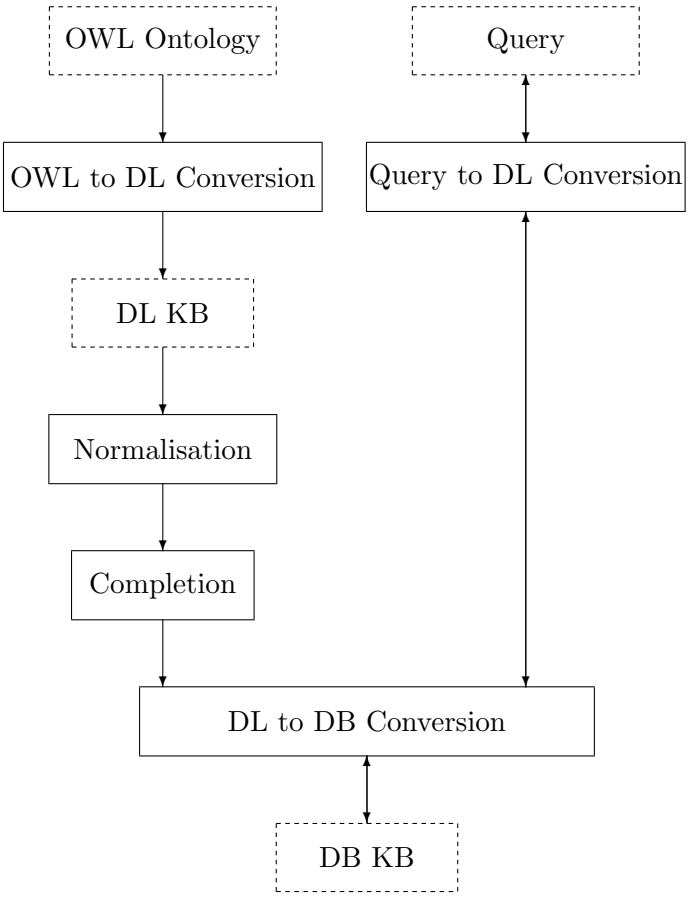

Figure 2.1: Load and query process.

In a first step, an OWL ontology will be converted to a description logic knowledge base O. Before calculating the extensions of the base concepts with the completion rules defined in Section 2.3.3, we have to normalise the knowledge base. Finally, a database instance has to be built from the completed ABox. The following sections will explain each conversion step in detail.

2.3.1 OWL to DL Conversion

As we have seen in Section 2.1, OWL offers a great flexibility in formalising information. Because OWL is based on description logics, the conversion from an OWL ontology into a description logic knowledge base is straightforward. There is no need to convert all possible OWL axioms because the defined description logic language does not support all possible

OWL axioms. The mapping we are using can be found in Table 2.1.

OWL-Axiom	**DL-Axiom**
Conversion for Classes	
`owl:Class`	Concept C
`owl:subClassOf`	Inclusion $\sqsubseteq$
`owl:equivalentClass`	Equivalence $\equiv$
`owl:intersectionOf`	Conjunction $\sqcap$
`owl:Thing`	Top Concept $\top$
`owl:Restriction`	*(see restrictions)*
Conversion for Individuals	
`rdf:Description`	Individual a
`rdf:type`	Concept Assertion $C(a)$
$\langle$`Property`$\rangle$	Role Assertion $R(a, b)$
Conversion for Properties	
`owl:ObjectProperty`	Role S
`owl:TransitiveProperty`	Transitive Role S and $S \in R^+$
`owl:inverseOf`	Inverse Role S^-
`rdf:subPropertyOf`	Role Inclusion $\sqsubseteq$
Conversion for Restrictions	
`owl:allValuesFrom`	Value Restriction $\forall R.C$
`owl:someValuesFrom`	Full Existential Quantification $\exists R.C$

Table 2.1: OWL and RDF to description logic mapping table.

The set of the mapped OWL axioms is a subset of OWL Lite [9], where the following axioms of OWL Lite are not included:

- `owl:equivalentProperty`, `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` and `owl:distinctMembers` because our defined description logic semantics respects the unique name assumption.

- `owl:DatatypeProperty`, `owl:SymmetricProperty`, `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`

- `owl:minCardinality`, `owl:maxCardinality` and `owl:cardinality`

- any annotation because they only contain additional information on an existing resource which is not relevant for the completion algorithm.

Furthermore, the defined description logic language does not contain the bottom concept. As a result, there is no mapping from the empty class `owl:Nothing` to the $\bot$ bottom concept.

Example

Listing 2.2 shows the OWL class *Carnivore* which is equivalent to an anonymous class. The anonymous class is a conjunction of the class *Animal* and a restriction. This restriction limits the set of individuals to those, which are related by the property *eats* to members of the class *Animal* only.

```
<owl:Class rdf:about="#Carnivore">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Animal">
        </owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#eats" />
          <owl:allValuesFrom>
            <owl:Class rdf:about="#Animal">
            </owl:Class>
          </owl:allValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Listing 2.2: Example of the ontology about species in OWL format which formalises the fact: "A carnivore is an animal which only eats Animals".

With the help of Table 2.1, we can map the OWL species example to the following description logic axiom:
$Carnivore \equiv Animal \sqcap \forall eats.Animal$
An equality whose left hand side is an atomic concept, is a concept definition. Therefore, the equivalence will be converted into a definition:
$Carnivore := Animal \sqcap \forall eats.Animal$

2.3.2 DL Normalisation

Before applying the completion rules, we need to convert O into a *normalised* knowledge base. The following iterative process describes the normalisation of an acyclic knowledge

base O:

1. Fully unfold the concept definitions of $O_{\mathcal{T}}$ by replacing each occurence of a name symbol on the right hand side of a definition with the concepts it stands for.
2. Replace concept descriptions of the form $\forall R.(C \sqcap D)$ by $\forall R.C \sqcap \forall R.D$.
3. Replace each concept assertion $C(a) \in O_{\mathcal{A}}$ by $C'(a)$ where $C'(a)$ is built from C by substituting each name symbol with its (unfolded and normalised) definition.

The normalised knowledge base contains only atomic concepts which are base symbols, because each atomic concept that has a definition has been replaced by its definition. Therefore, there are only base symbols in all concept descriptions and thus, in all concept assertions.

Example

Assume that the knowledge base O is given by:
$O_{\mathcal{T}} := \{Carnivore := Animal \sqcap \forall eats.Animal\}$ and
$O_{\mathcal{A}} := \{Carnivore(lion)\}$
After appling the normalisation rules on the knowledge base, we will get:
$O_{\mathcal{T}} := \{Carnivore := Animal \sqcap \forall eats.Animal\}$ and
$O_{\mathcal{A}} := \{Animal \sqcap \forall eats.Animal(lion)\}$

2.3.3 DL Completion

The most important step in the loading process is the completion procedure. The aim of the completion is to compute all atomic concept and role assertions which are entailed by the knowlege base O. For this purpose, we define a new relation $O \vdash s$ where s is an ABox assertion and means that "s follows from O":

(R1) $O \vdash C(a)$ if $C(a) \in O$

(R2) $O \vdash R(a, b)$ if $R(a, b) \in O$

(R3) $O \vdash \top(a)$, for all individuals $a \in O$

(R4) $O \vdash C(a)$ if $O \vdash C \sqcap D(a)$ or $O \vdash D \sqcap C(a)$

(R5) $O \vdash C(x)$ and $O \vdash R(a, x)$ and $O \vdash \top(x)$ if $O \vdash \exists R.C(a)$, (x new constant)

(R6) $O \vdash C(a)$ if $\exists b \in O$ such that $O \vdash \forall R.C(b)$ and $O \vdash R(b, a)$

(R7) $O \vdash C(\forall_{R.a})$ and $O \vdash \top(\forall_{R.a})$ if $O \vdash \forall R.C(a)$

(R8) $O \vdash C(a)$ if $O \vdash D(a)$ and $D \sqsubseteq C \in O$

(R9) $O \vdash T(a, b)$ if $O \vdash S(a, b)$ and $S \sqsubseteq T \in O$

(R10) $O \vdash S^{-}(b, a)$ if $O \vdash S(a, b)$

(R11) $O \vdash S(a, c)$ if $O \vdash S(a, b)$ and $O \vdash S(b, c)$ and $S \in R^{+}$

We complete the knowledge base O by iterating through rules R1 to R11 and applying each rule until no new consequences occur. Let's have a look at the diffrent rules briefly:

- R1 and R2 treat the assertions which are already included in O.
- R3 adds a top concept assertion for all indviduals in O since they are all members of the domain of interpretation.
- R4 and R6 compute the immediate consequences of concept assertions of the form $C \sqcap D(a)$ and $\forall R.C(b)$.
- R5 deals with full existential quantification. The consequence of an assertion of the form $\exists R.C(a)$ is that there must exist at least one individual x in the concept C such that a is related to x by the role R. R5 adds a new constant x as a witness. We have to ensure that there is only one new constant added for each distinct full existential quantification assertion. However, there is always a constant added even if there already exists a witness in the knowledge base.
- R7 adds a special constant $\forall_{R.a}$ belonging to concept C for every assertion of the form $\forall R.C(a)$. The purpose of this constant will be explained in Section 2.3.4.
- R8 and R9 compute assertions following directly from concept and role hierarchy axioms.
- R10 and R11 add new role assertions for role assertions which contain inverse or transitive roles.

The next example will show how these completion rules are applied to our species knowledge base.

Example

Assume that the acyclic knowledge base O is already normalised and given by:

TBox $O_{\mathcal{T}}$**:**

(T1) $Carnivore := Animal \sqcap \forall eats.Animal$

(T2) $Omnivore := Animal \sqcap \exists eats.Animal \sqcap \exists eats.Plant$

(T3) $Animal \sqsubseteq Creature$

(T4) $Plant \sqsubseteq Creature$

(T5) $eats \sqsubseteq hunts$

Name symbols $N_{\mathcal{T}} := \{Carnivore, Omnivore\}$
Base symbols $B_{\mathcal{T}} := \{Creature, Animal, Plant\}$

ABox $O_{\mathcal{A}}$**:**

(A1) $\top(zebra)$

(A2) $eats(lion, zebra)$

(A3) $Animal \sqcap \forall eats.Animal(lion)$

(A4) $Animal \sqcap \exists eats.Animal \sqcap \exists eats.Plant(bear)$

Note that (A3) is the normalised assertion $Carnivore(lion)$ and (A4) is equal to the concept assertion $Omnivore(bear)$. Let us compute all consequences which can be found by applying the completion rules to the knowledge base O. All new assertions are grouped by the rule they are created from and marked on the right hand side with the assertion numbers used to compute the consequence.

Rule 1:

(1) $O \vdash \top(zebra)$ (A1)

(2) $O \vdash Animal \sqcap \forall eats.Animal(lion)$ (A3)

(3) $O \vdash Animal \sqcap \exists eats.Animal \sqcap \exists eats.Plant(bear)$ (A4)

Rule 2:

(4) $O \vdash eats(lion, zebra)$ (A2)

Rule 3:

(5) $O \vdash \top(lion)$

(6) $O \vdash \top(bear)$

Rule 4:

(7) $O \vdash Animal(lion)$ (2)

(8) $O \vdash \forall eats.Animal(lion)$ (2)

(9) $O \vdash Animal(bear)$ (3)

(10) $O \vdash \exists eats.Animal(bear)$ (3)

(11) $O \vdash \exists eats.Plant(bear)$ (3)

Rule 5:

(12) $O \vdash Animal(a)$ (10)

(13) $O \vdash eats(bear, a)$ (10)

(14) $O \vdash \top(a)$ (10)

(15) $O \vdash Plant(b)$ (11)

(16) $O \vdash eats(bear, b)$ (11)

(17) $O \vdash \top(b)$ (11)

Rule 6:

(18) $O \vdash Animal(zebra)$ (8 + 4)

Rule 7:

(19) $O \vdash Animal(\forall_{eats.lion})$ (8)

(20) $O \vdash \top(\forall_{eats.lion})$ (8)

Rule 8:

(21) $O \vdash Creature(lion)$ (7 + T3)

(22) $O \vdash Creature(bear)$ (9 + T3)

(23) $O \vdash Creature(a)$ (12 + T3)

(24) $O \vdash Creature(b)$ (15 + T4)

Rule 9:

(25) $O \vdash hunts(lion, zebra)$ (4 + T5)

(26) $O \vdash hunts(bear, a)$ (13 + T5)

(27) $O \vdash hunts(bear, b)$ (16 + T5)

This example shows already that even for a small knowledge base (4 assertions, 2 individuals) we can compute quite a few consequences (27 assertions, 5 individuals).

2.3.4 DL to DB Conversion

Finally, the completed knowledge base O has to be converted into a database. Therefore, we introduce a relational database instance $\widehat{O}$ and define it by:

- $A(a)$ is included in $\widehat{O}$ if $O \vdash A(a)$ for a base symbol A
- $R(a, b)$ is included in $\widehat{O}$ if $O \vdash R(a, b)$

Hence, the newly created database instance contains all atomic concept and role assertions which are consequences of the knowledge base O. Note that complex concept assertions like value restrictions are not included in $\widehat{O}$. However, we need a mechanism to store complex assertions, specifically concept definitions, in the database as well.

Let us introduce a special consequence relation $\widehat{O} \models_{\mathsf{DB}} C(a)$ for the database instance $\widehat{O}$ and a concept descripton C, which means that "a belongs to C follows from the database instance $\widehat{O}$". Thus, we are now able to add complex concept definitions to our database.

$A(a)$ of a concept definition $A := C \in O$ is included in $\widehat{O}$ if $\widehat{O} \models_{\mathsf{DB}} C(a)$, whereas $\widehat{O} \models_{\mathsf{DB}} C(a)$ is inductively defined by:

1. $\widehat{O} \models_{\mathsf{DB}} A(a)$ if $A(a) \in \widehat{O}$
2. $\widehat{O} \models_{\mathsf{DB}} R(a, b)$ if $R(a, b) \in \widehat{O}$
3. $\widehat{O} \models_{\mathsf{DB}} C \sqcap D(a)$ if $\widehat{O} \models_{\mathsf{DB}} C(a)$ and $\widehat{O} \models_{\mathsf{DB}} D(a)$
4. $\widehat{O} \models_{\mathsf{DB}} \forall R.C(a)$ if $\widehat{O} \models_{\mathsf{DB}} C(\forall_{R.a})$
5. $\widehat{O} \models_{\mathsf{DB}} \exists R.C(a)$ if $\exists b \in \widehat{O}$ such that $\widehat{O} \models_{\mathsf{DB}} R(a, b)$ and $\widehat{O} \models_{\mathsf{DB}} C(b)$

The previous definition gives us a translation of logic retrieval queries (complex definitions) to relational database queries. With the help of the introduced special constant $\forall_{R.a}$, we can answer a complex query of the form $\forall R.C(a)$ simply by checking if the individual $\forall_{R.a}$ belongs to concept C (rule 4).

The next example will show, how the database instance is built.

Example

Let the knowledge base O be given by the TBox $O_{\mathcal{T}} := \{Carnivore := Animal \sqcap \forall eats.Animal\}$ and the following consequences:

- $O \vdash Animal(lion)$
- $O \vdash Animal(\forall_{eats.lion})$
- $O \vdash eats(lion, zebra)$
- $O \vdash Animal \sqcap \forall eats.Animal(lion)$

In a first conversion step, all atomic concept and role assertions are added to the database instance $\widehat{O}$:

- $Animal(lion)$
- $Animal(\forall_{eats.lion})$
- $eats(lion, zebra)$

Finally, the assertions for the defined concept $Carnivore$ are added to the database instance by applying the database consequence relation definition:

$Carnivore(a)$ is added to $\widehat{O}$ if $\widehat{O} \models_{\mathsf{DB}} Animal \sqcap \forall eats.Animal(a)$. Hence, $Animal \sqcap \forall eats.Animal(a)$ follows only from $\widehat{O}$ if $\widehat{O} \models_{\mathsf{DB}} Animal(a)$ and $\widehat{O} \models_{\mathsf{DB}} Animal(\forall_{eats.a})$. The last consequences only contain atomic concept assertions. Thus, $\widehat{O} \models_{\mathsf{DB}} Animal(a)$ if $Animal(a) \in \widehat{O}$ and $\widehat{O} \models_{\mathsf{DB}} Animal(\forall_{eats.a})$ if $Animal(\forall_{eats.a}) \in \widehat{O}$. The individual $lion$ satisfies these conditions and a new concept assertion built out of the definition of $Carnivore$ is added:

- $Carnivore(lion)$

3 Implementation

After defining the theoretical basics of the knowledge base, we are now able to look at the concrete implementation in detail. Section 3.1 describes briefly all packages of the program and their dependencies. Additionally, a list of the used external libraries and the database specification are presented. The most important algorithms, the completion and the description logic to database conversion, are then explained in Section 3.2. The program is called REAL and is written in Java[1] (Version 1.4.2).

The attached CD contains all the source code of REAL as well as binaries of the used libraries. The API is well documented and the documentation can be found in the `doc` directory. Furthermore, there is an installation instruction file (`INSTALL`) which describes all necessary steps to run REAL. See Appendix A.1 for a listing of the directory tree on the CD.

3.1 Design

3.1.1 Packages

In this section, I will discuss each package of REAL briefly. Figure 3.1 shows all packages and their most important classes along with their dependencies.

ch.unibe.til.real.db

The database package is responsible for database specific operations such as managing connections to the database, initialising indexes or deleting table entries. This functionality is abstracted in the class `DBKnowledgeBase`. Note that table creation is done using a SQL script[2], which needs to be executed once in advance. All database depending constants, such as table names, are defined separately (in the interface `DBConstants`).

[1] `http://java.sun.com`
[2] The SQL table creation script can be found in the `scripts` directory.

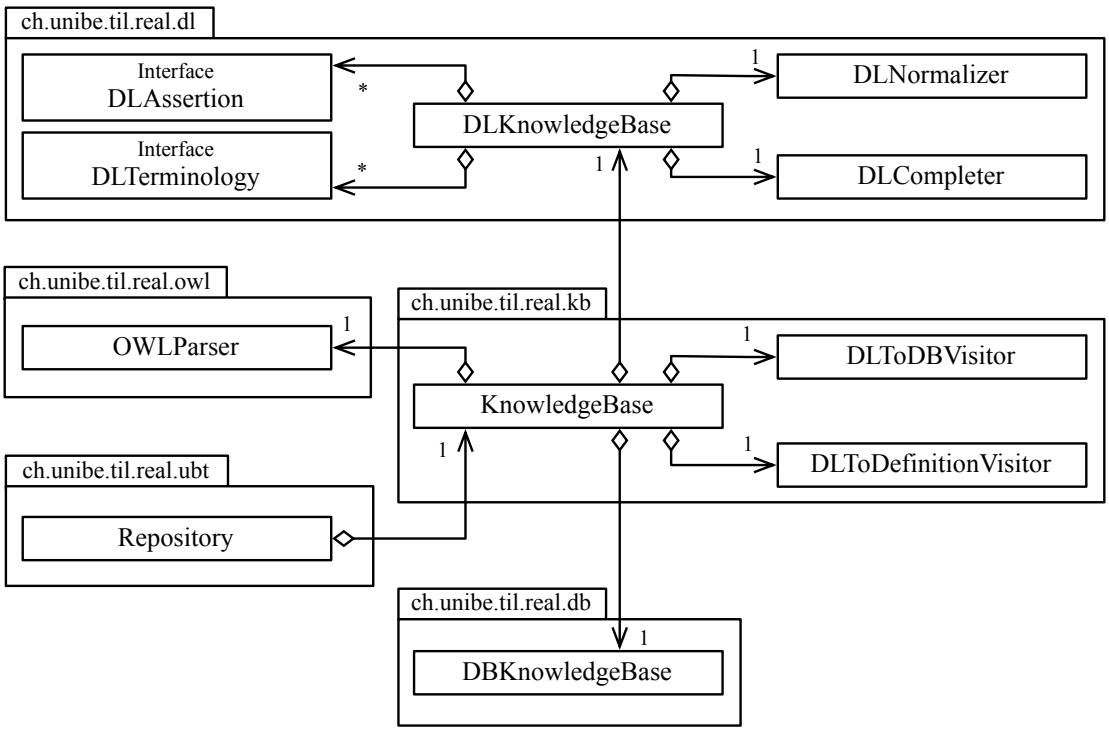

Figure 3.1: UML class diagram with packages of the core classes.

ch.unibe.til.real.dl

The package `dl` contains all description logic related data and operations. The syntax and semantics introduced in Section 2.2 are abstracted directly into objects. The UML class diagram in Figure 3.2 shows the dependencies between those objects.

All description logic objects implement the interface `DLObject` and generalise the abstract class `DLAbstractObject`. There are two other important interfaces, `DLAssertion` and `DLTerminology`. The former is realised by `DLConceptAssertion` and `DLRoleAssertion`, whereas the latter is the interface for `DLRoleInclusion`, `DLInclusion` (concept inclusion) and `DLDefinition`. The class `DLExtendedIndividual` encapsulates the special constant $\forall_{R.a}$ and extends the default individual class `DLIndividual`. Therefore, the `DLExtendedIndividual` class contains the role R and the individual a of $\forall_{R.a}$. Note that the extended individual itself can contain an extended individual which then stands for $\forall_{R.\forall_{R.a}}$.

The class `DLKnowledgeBase` integrates the TBox and ABox, thus it contains all as-

sertion and terminology objects (`DLAssertion`, `DLTerminology`). Furthermore, the class implements the factory pattern [6] to ensure that only one instance of the same object is created. Therefore, all other classes have to call `DLKnowledgeBase` for description logic instance creation.

The normalisation (presented in Section 2.3.2) of a `DLKnowledgeBase` is achieved by the `DLNormalizeVisitor` and, as the name states, the class uses the visitor pattern found in [6] for normalisation. This means that only the visitor knows how to normalise a description logic object.

Finally, the class `DLCompleter` belongs to the `dl` package. This class realises all completion rules defined in Section 2.3.3. Section 3.2.1 describes the implemented algorithm in detail.

ch.unibe.til.real.kb

The knowledge base package is the core package. The class `KnowledgeBase` is responsible for the whole load and query process described in Section 2.3 and acts as a controller. This knowledge base object contains instances of `OWLParser`, `DLKnowledgeBase` and `DBKnowledgeBase`. Furthermore, there are two visitor classes (`DLtoDBVisitor` and `DLDefinitionToDBVisitor`), which carry out the database conversion of the description logic knowledge base as defined in 2.3.4. The actual implementation of theses visitors is described in Section 3.2.2.

ch.unibe.til.real.owl

The class `OWLParser` in the `owl` package is responsible for parsing the OWL files. The actual parsing is done by the `OWLParserHandler` class, which is an implementation of the SAX `ContentHandler` interface (see Section 3.1.2).

ch.unibe.til.real.ubt

The classes of the package `ubt` implement the interface of the UBT Benchmark (see Section 4.1.1). The class `Repository` controls the benchmark since it delegates knowledge base load and query calls from UBT to the `KnowledegeBase` class. There exists no main method in REAL and the `Repository` class is the only possibility to load data into the knowledge base or to issue queries on the data.

Queries have to be formulated directly in SQL syntax and are passed to the method

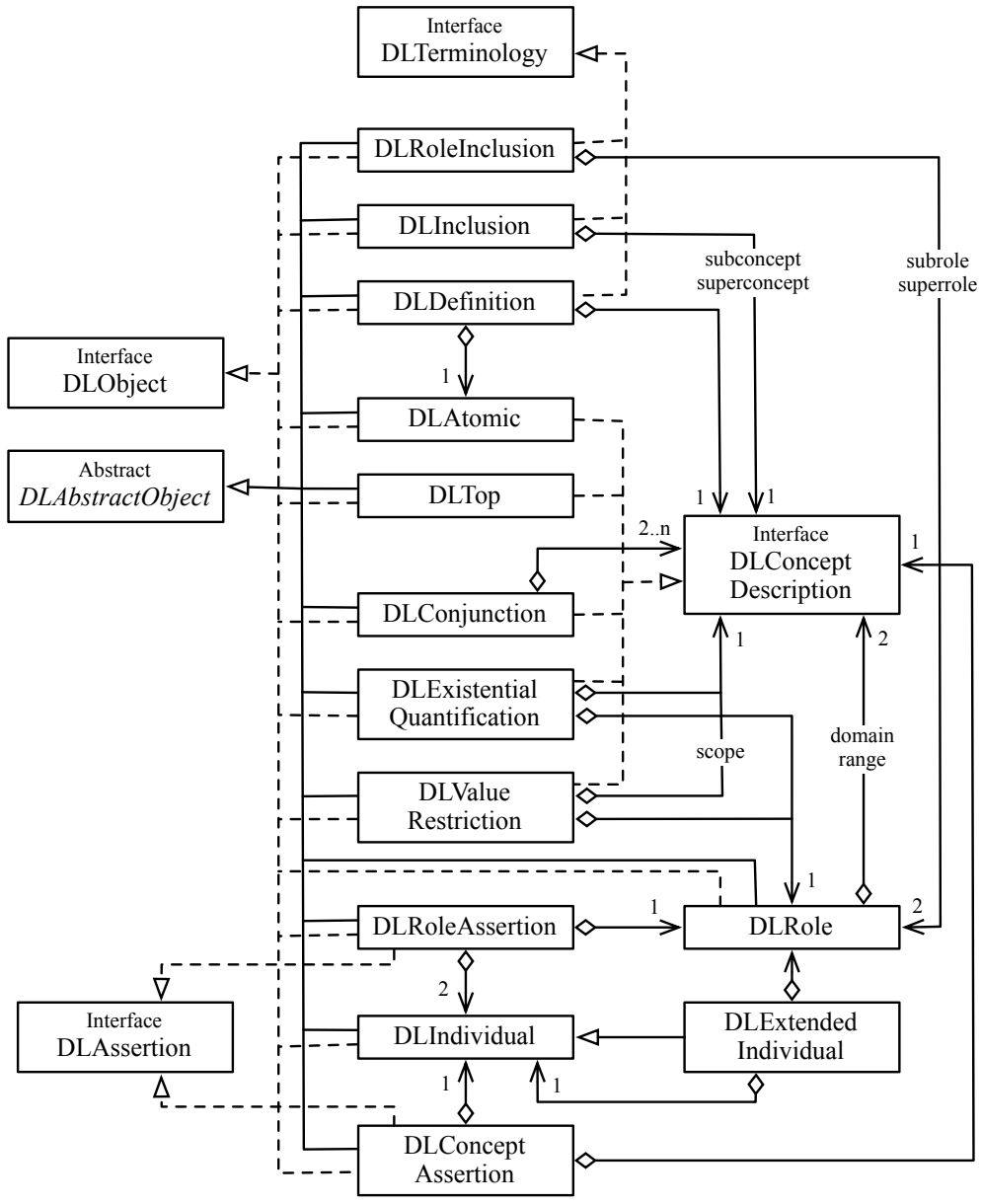

Figure 3.2: UML class diagram of the `ch.unibe.til.real.dl` description logic package.

`issueQuery` in the class `Repository`. The method returns itself an object `QueryResult` which contains a Java SQL query `ResultSet` object.

ch.unibe.til.real.*.test

The packages `dl`, `kb` and `owl` contain a package `test` with unit tests.

3.1.2 External Libraries

PostgreSQL JDBC Driver The PostgreSQL JDBC Driver[3] (Version 8.0-312) allows REAL to connect to a PostgreSQL server using Java's database independent JDBC programming interface[4].

SAX XML Parser SAX[5] denotes "Simple API for XML"and is a common interface implemented for many different XML parsers. A SAX implementation is already included in Java (Version 1.4.2). SAX allows to traverse through XML files sequentially. Thus, the elements are read one by one, each access triggering an event. Therefore, SAX is useful for processing very large XML documents because it does not model a tree structure like DOM (Document Object Model), which uses a lot of runtime memory. Since OWL files are based on XML syntax, we can use the SAX interface for parsing ontology files efficiently.

Log4j Logging Service Log4j[6] (Version 1.2.13) is a fully configurable log API. It supports log level changes at runtime which proves very useful for debugging.

3.1.3 Database

The database management system (DBMS) is an important part of the implementation. We need a DBMS with which we are able to change server parameters and to analyse query execution plans in order to optimise query answering time. PostgreSQL[7] (Version 8.1.3) meets these criterias and is available under the BSD open source license[8].

[3] `http://jdbc.postgresql.org/`
[4] `http://java.sun.com/products/jdbc/`
[5] `http://www.saxproject.org/`
[6] `http://logging.apache.org/`
[7] `http://www.postgresql.org/`
[8] `http://www.postgresql.org/about/licence`

Database Schema

The chosen design of the database (see Figure 3.3) derives directly from the description logic model. Note that each table name has a unique prefix, which is also included in the attribute names. The description logic to database conversion in Section 2.3.4 does only add concept and role assertions to the database. Thus, there exist tables for concept assertions (`ca_concept_assertions`) and role assertions (`ra_role_assertions`). Furthermore, all individuals are stored in the table `i_individuals`. Hence, each role assertion references to two individuals (domain and role individual) whereas each concept assertion contains one individual as a foreign key.

Additionally, an individual has a boolean attribute `i_constant` which marks an individual as an added constant. For example the rules R5 and R7 of the completion algorithm in Section 2.3.3 add new individuals. These added constants are only used by the description logic to database conversion algorithm (Section 2.3.4). Therefore, the constant attribute helps to exclude these individuals in a query result.

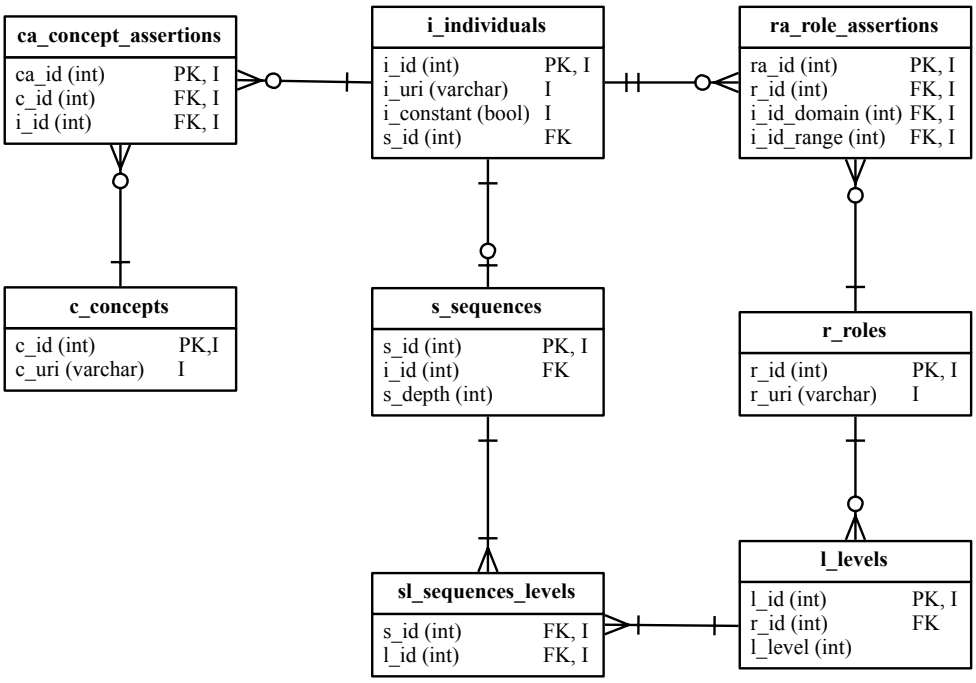

Figure 3.3: Database UML diagram where PK denotes primary key, FK stands for foreign key and I marks the existence of an index on the corresponding attribute.

The Individual $\forall_{R.a}$

A problem appears in the context of the extended individuals. $\forall_{R.a}$ itself can contain a finite chain of extended individuals ($\forall_{R.\ldots.\forall_{R.a}}$). The tables `s_sequences`, `l_levels` and `sl_sequences_levels` are used to store such individuals.

If the foreign key `s_id` in the table `i_individuals` has a value other than `NULL`, the individual is an extended individual with the sequence `s_id`. A sequence denotes a whole chain of $\forall_{R.a}$. It has a depth (`s_depth`) which marks the number of chained individuals. For example $\forall_{R.a}$ has depth 1, $\forall_{R.\forall_{R.a}}$ has depth 2, $\forall_{R.\forall_{R.\forall_{R.a}}}$ has depth 3 and so on.

Each sequence has different levels with roles. For example the individual $\forall_{R.\forall_{S.a}}$ of depth 2 contains at level 1 the role R and at level 2 the role S. These levels are modeled by the table `l_level`. Each level has a level number (`l_levels`) and a foreign key `r_id` which is related to a role. The table `sl_sequences_levels` is introduced to solve the many-to-many relation between sequences and levels.

A sequence has also a final individual (`i_id` in table `s_sequences`). This foreign key references to the individual, which is the last non extended individual in the chain $\forall_{R.a}$, here for example a.

The detailed conversion of an extended individual into a database entry, as well as a example, will be given in Section 3.2.2.

3.2 Algorithms

3.2.1 Completion

The completion rules defined in Section 2.3.3 are coded in the class `DLCompleter` contained in the package `dl`. Each rule is implemented directly as a separate method. However, the important part of the class is the call sequence of each method (listed in Algorithm 3.1) to improve completion performance. Furthermore, there are three helper knowledge bases ($\bar{O}$, O' and $\tilde{O}$) introduced to minimise the number of assertions in the input set of each rule.

3.2.2 DL To Database Conversion

The description logic to database conversion introduced in Section 2.3.4 already defines all necessary rules to store a description logic knowledge base into a database. The database scheme (Figure 3.3) is also given in Section 3.1.3. Therefore, the only task remaining is

Input: Knowledge Base O
Output: Completed Knowledge Base $\bar{O}$

$\bar{O} = \{\}$; $O' = \{\}$; $\tilde{O} = \{\}$ // Initialise helper knowledge bases
$\bar{O} \vdash C(a)$ if $C(a) \in O$ // Rule 1
$\bar{O} \vdash R(a,b)$ if $R(a,b) \in O$ // Rule 2
$\bar{O} \vdash \top(a)$, for all $a \in O$ // Rule 3
$\tilde{O} = \bar{O}$
repeat // apply rules 4 to 11 on consequences computed by inclusion rules 8 and 9
  $O' = \tilde{O}$
  repeat
    $assertionAdded =$ **false**
    if $(O' \vdash C \sqcap D(a)$ or $O' \vdash D \sqcap C(a))$ and $O' \nvdash C(a)$ **then** // Rule 4
      $O' \vdash C(a)$
      $assertionAdded =$ **true**
    end if
    if $O' \vdash \exists R.C(a)$ and $(O' \nvdash C(b)$ and $O' \nvdash R(a,b)$ and $O' \nvdash \top(b))$ **then** // Rule 5
      $O' \vdash C(b)$ and $O' \vdash R(a,b)$ and $O' \vdash \top(b)$ (b new constant)
      $assertionAdded =$ **true**
    end if
    if $\exists b \in O$ such that $O' \vdash \forall R.C(b)$ and $O' \vdash R(b,a)$ and $O' \nvdash C(a)$ **then** // Rule 6
      $O' \vdash C(a)$
      $assertionAdded =$ **true**
    end if
    if $O' \vdash \forall R.C(a)$ and $O' \nvdash C(\forall_{R.a})$ and $O' \nvdash \top(\forall_{R.a})$ **then** // Rule 7
      $O' \vdash C(\forall_{R.a})$ and $O' \vdash \top(\forall_{R.a})$
      $assertionAdded =$ **true**
    end if
    if $O' \vdash S(a,b)$ and $O' \nvdash S^{-}(b,a)$ **then** // Rule 10
      $O' \vdash S^{-}(b,a)$
      $assertionAdded =$ **true**
    end if
    if $O' \vdash S(a,b)$ and $\bar{O} \vdash S(b,c)$ and $S \in R^{+}$ and $O' \nvdash S(a,c)$ and $\bar{O} \nvdash S(a,c)$ **then** // Rule 11
      $O' \vdash S(a,c)$
      $assertionAdded =$ **true**
    end if
  until $assertionAdded =$ **false**
  $\tilde{O} = \{\}$
  if $O' \vdash D(a)$ and $D \sqsubseteq C \in O$ and $\tilde{O} \nvdash C(a)$ **then** // Rule 8
    $\tilde{O} \vdash C(a)$
  end if
  if $O' \vdash S(a,b)$ and $S \sqsubseteq T \in O$ and $\tilde{O} \nvdash T(a,b)$ **then** // Rule 9
    $\tilde{O} \vdash T(a,b)$
  end if
  $\bar{O} = \bar{O} \cup O'$
until $\tilde{O} = \{\}$
return $\bar{O}$

Algorithm 3.1: Completion Algorithm

to convert the rules to SQL statements which insert the data into the database. Like in Section 2.3.4, the conversion is divided into two steps, an initial conversion and a definition conversion.

Initial Conversion

The initial conversion is equivalent to the definition of the database instance $\widehat{O}$. Only atomic concept assertions and role assertions have to be added to the database during this step.

Assume the database instance $\widehat{O}$ is already given and contains all atomic concept and role assertions produced by the completion. The pseudocode of the conversion *initialConvert* is shown in Algorithm 3.2.

The method *nextval(tableprefix_seq)* used in all algorithms returns the next free sequence number of the table with the given prefix. This sequence number is treated as a unique id, which is the primary key of the table. Additionally, the *execute(SQL_Statement)* method will execute the parameter SQL statement.

Input: Database instance $\widehat{O}$

for all $x \in \widehat{O}$ **do**
  if x is an atomic concept assertion of the form $A(a)$ **then**
    // Call individual and atomic insert methods and insert concept assertion.
    $ca\_id = nextval('ca\_seq')$
    $c\_id = insertAtomicConcept(A)$
    $i\_id = insertIndividual(a)$
    execute(`INSERT INTO ca_concept_assertions (ca_id, c_id, i_id)`
      `VALUES (`*ca_id, c_id, i_id*`)`*)*
  end if

  if x is a role assertion of the form $R(b, c)$ **then**
    // Call role and both individual insert methods and insert role assertion.
    $ra\_id = nextval('ra\_seq')$
    $r\_id = insertRole(R)$
    $i\_id\_domain = insertIndividual(b)$
    $i\_id\_range = insertIndividual(b)$
    execute(`INSERT INTO ra_role_assertions (ra_id, r_id, i_id_domain, i_id_range)`
      `VALUES (`*ra_id, r_id, i_id_domain, i_id_range*`)`*)*
  end if
end for

Algorithm 3.2: initialConvert($\widehat{O}$)

The methods *insertAtomicConcept()* (Algorithm 3.3), *insertRole()* (Algorithm 3.4) and

insertIndividual() (Algorithm 3.5) are listed separately to increase readability. They are called before a role or an atomic concept assertion is added to the database in order to get the foreign keys.

Let us have a closer look at the Algorithm 3.5 *insertIndividual()*. It contains a case distinction on the type of the individual. In case of an extended individual of the form $\forall_{R.a}$, a loop writes all levels and the corresponding rules into the database until the whole extended individual is unfolded. Moreover, the individual at the end of the chain, named $i2\_id$, is also included into the database. Finally, the sequence, its depth and the reference to the last individual $i2\_id$ is written into the table `s_sequences`.

Input: Atomic concept A
Output: Database id c_id of the inserted concept

```
c_id = nextval('c_seq')
execute( INSERT INTO c_concepts (c_id, c_uri)
        VALUES (c_id, 'A'))
return c_id
```

Algorithm 3.3: insertAtomicConcept(A)

Input: Role R
Output: Database id r_id of the inserted role

```
r_id = nextval('r_seq')
execute( INSERT INTO r_roles (r_id, r_uri)
        VALUES (r_id, 'R'))
return r_id
```

Algorithm 3.4: insertRole(R)

Definition Conversion

In a second step, all individuals belonging to a complex concept definition are added to the database. Remember from Section 2.3.4 that $A(a)$ of a concept definition $A := C$ is included in the database instance $\widehat{O}$ if $\widehat{O} \models_{\mathsf{DB}} C(a)$. All we have to do is collecting all individuals which are members of C and insert them into the database as concept assertions with the atomic concept A. This can be done in one SQL statement because SQL allows the combination of `INSERT` and `SELECT` commands. Algorithm 3.6 shows how these statements are combined.

Furthermore, the atomic concept A has to be included in the database as well. This is done by calling the method *insertAtomicConcept(A)*. The `SELECT` command construction,

Input: Individual x
Output: Database id i_id of the inserted individual

```
i_id = nextval('i_seq')
if x is an extended individual of the form ∀_{R.a} then
  s_id = nextval('s_seq')
  level = 1
  repeat
    // Insert levels and roles of the extended individual.
    l_id = nextval('l_seq')
    r_id = insertRole(R)
    execute(INSERT INTO l_levels (l_id, r_id, l_level)
            VALUES (l_id, r_id, level)
    execute(INSERT INTO sl_sequences_levels (s_id, l_id)
            VALUES (s_id, l_id))
    level = level + 1
    x = a
  until x is not an extended individual
  // Insert sequence and last individual of sequence.
  i2_id = insertIndividual(x)
  execute(INSERT INTO s_sequences (s_id, i_id, s_depth)
          VALUES (s_id, i2_id, level))
  // Insert extended individual as a constant with a sequence.
  execute(INSERT INTO i_individuals (i_id, i_uri, i_constant, s_id)
          VALUES (i_id, '∀_{...R.x}', true, s_id))

else if x is an added constant then
  // Insert individual as a constant with no sequence.
  execute(INSERT INTO i_individuals (i_id, i_uri, i_constant, s_id)
          VALUES (i_id, 'x', true, NULL))

else
  // Insert individual which is no constant and has no sequence.
  execute(INSERT INTO i_individuals (i_id, i_uri, i_constant, s_id)
          VALUES (i_id, 'x', false, NULL))
end if
return i_id
```

Algorithm 3.5: insertIndividual(x)

which returns all members of the concept C, is listed separately in Algorithm 3.7. This construction method may be called recursivly, for example in case of a conjunction.

On the other hand, if C is a value restriction $\forall R.D$, the `SELECT` statement is complex, as we have to choose those individuals a where $\forall_{R.a}$ is a member of the concept D. Therefore, the query includes the tables `s_sequences`, `sl_sequences_levels` and `l_levels` for building the extended individual hierarchy. The statement gets even more complex if the concept D is also a value restriction. Hence, the `SELECT` statement for value restrictions is built using a loop over the depth of the extended individual.

Input: Knowledge Base O

for all concept definitions $A := C \in O$ **do**
  c_id = *insertAtomicConcept*(A)
  statement =
    `INSERT INTO ca_concept_assertions (ca_id, c_id, i_id)`
    `SELECT DISTINCT nextval('ca_seq'),` *c_id*`, i_id`
    `FROM i_individuals`
    `WHERE i_id IN (`*createSelectStatement*(C)`)`
  execute(statement)
end for

Algorithm 3.6: definitionConvert(O)

Example

Let the knowledge base O be given by the TBox $O_{\mathcal{T}} := \{Carnivore := Animal \sqcap \forall eats.Animal\}$ and the database instance $\widehat{O} := \{Animal(lion), Animal(\forall_{eats.lion})\}$.

First, we execute the initial conversion Algorithm 3.2 called *initConvert($\widehat{O}$)* with input $\widehat{O}$, which iterates over all members of $\widehat{O}$ and results in the following SQL statements:

For $Animal(lion)$:

```
INSERT INTO c_concepts (c_id, c_uri)
VALUES (1,'Animal')

INSERT INTO i_individuals (i_id, i_uri, i_constant, s_id)
VALUES (1,'lion', false, NULL)

INSERT INTO ca_concept_description (ca_id, c_id, i_id)
VALUES (1, 1, 1)
```

For $Animal(\forall_{eats.lion})$:

```
INSERT INTO r_roles (r_id, r_uri)
VALUES (1,'eats')
```

Input: Concept description C
Output: SQL statement which selects all individuals belonging to concept C

if C is an atomic concept of the form A **then**
  statement = `SELECT DISTINCT i.i_id`
    `FROM i_individuals AS i, c_concepts AS c, ca_concept_assertions AS ca`
    `WHERE ca.i_id = i.i_id AND ca.c_id = c.c_id AND`
      `c.c_uri = 'A'`
end if

if C is a conjunction of the form $D \sqcap E$ **then**
  statement =(*createSelectStatement*(D)) `INTERSECT` (*createSelectStatement*(E))
end if

if C is a value restriction of the from $\forall R.D$ **then**
  depth = 0
  statement =`SELECT DISTINCT i1.i_id`
  fromStatement =`FROM`
  whereStatement =`WHERE`
  repeat
    depth = *depth* + 1
    fromStatement = *fromStatement*+ `l_levels AS l`*depth*`, r_roles AS r`*depth*`,`
    whereStatement = *whereStatement*+
      `sl.l_id = l`*depth*`.l_id AND`
      `l`*depth*`.r_id = r`*depth*`.r_id AND`
      `l`*depth*`.l_level =` *depth* `AND`
      `r`*depth*`.r_uri = 'R' AND`
    $C = D$
  until C is not a value restriction of the form $\forall R.D$
  fromStatement = *fromStatement*+
    `i_individuals AS i1, i_individuals AS i2,`
    `s_sequences AS s, sl_sequences_levels AS sl`
  whereStatement = *whereStatement*+
    `i1.i_id = s.i_id AND s.s_id = sl.s_id AND`
    `s.s_depth =` *depth* `AND`
    `i2.s_id = s.s_id AND`
    `i2.i_id IN (`*createSelectStatement*(D)`)`
  statement = *fromStatement* + *whereStatement*
end if

if C is an existential quantification of the from $\exists R.D$ **then**
  statement = `SELECT DISTINCT idomain.i_id`
    `FROM i_individuals AS idomain, i_individuals AS irange,`
      `r_roles AS r, ra_role_assertions AS ra`
    `WHERE ra.i_id_domain = idomain.i_id AND`
      `ra.i_id_range = irange.i_id AND`
      `ra.r_id = r.r_id AND`
      `r.r_uri = 'R' AND`
      `i2.i_id IN (`*createSelectStatement*(D)`)`
end if
return *statement*

Algorithm 3.7: createSelectStatement(C)

```
INSERT INTO l_levels (l_id, r_id, l_level)
VALUES (1, 1, 1)
INSERT INTO sl_sequences_levels (s_id, l_id)
VALUES (1, 1)
INSERT INTO s_sequences (s_id, i_id, s_depth)
VALUES (1, 1, 1)
INSERT INTO i_individuals (i_id, i_uri, i_constant, s_id)
VALUES (2,'∀eats,lion', true, 1)
INSERT INTO ca_concept_descriptions (ca_id, c_id, i_id)
VALUES (2, 1, 2)
```

Hence, the resulting tables of the database are:

ca_concept_assertions

ca_id	c_id	i_id
1	1	1
2	1	2

i_individuals

i_id	i_uri	i_constant	s_id
1	*lion*	false	NULL
2	$\forall_{eats.lion}$	true	1

r_roles

r_id	r_uri
1	*eats*

c_concepts

c_id	c_uri
1	*Animal*

s_sequences

s_id	i_id	s_depth
1	1	1

l_levels

l_id	l_level	r_id
1	1	1

sl_sequences_levels

s_id	l_id
1	1

The definition conversion of the knowledge base O and its definition $Carnivore := Animal \sqcap \forall eats.Animal$ by the Algorithm 3.6 results in the following SQL call:

```
INSERT INTO ca_concept_assertions (ca_id, c_id, i_id)
SELECT DISTINCT nextval('ca_seq'), 1, i_id
FROM   i_individuals
WHERE  i_id IN (
  (SELECT DISTINCT i.i_id
   FROM   i_individuals AS i, c_concepts AS c, ca_concept_assertions AS ca
   WHERE  ca.i_id = i.i_id AND
          ca.c_id = c.c_id AND
          c.c_uri = 'Animal')
  INTERSECT
  (SELECT DISTINCT i1.i_id
   FROM   l_levels AS l1, r_roles AS r1,
          i_individuals AS i1, i_individuals AS i2,
          s_sequences AS s, sl_sequences_levels AS sl
```

```
 WHERE  sl.l_id = l1.l_id AND
        l1.r_id = r1.r_id AND
        l1.l_level = 1 AND
        r1.r_uri = 'eats' AND
        i1.i_id = s.i_id AND
        s.s_id = sl.s_id AND
        s.s_depth = 1 AND
        i2.s_id = s.s_id AND
        i2.i_id IN (
          SELECT DISTINCT i.i_id
          FROM   i_individuals AS i, c_concepts AS c, ca_concept_assertions AS ca
          WHERE  ca.i_id = i.i_id AND
          ca.c_id = c.c_id AND
          c.c_uri = 'Animal')
)
```

Altogether, a new concept *Carnivore* and concept assertion *Carnivore*(*lion*) are added to the database (bold) and the tables now look as follows:

ca_concept_assertions

ca_id	c_id	i_id
1	1	1
2	1	2
3	**2**	**1**

c_concepts

c_id	c_uri
1	*Animal*
2	**Carnivore**

i_individuals

i_id	i_uri	i_constant	s_id
1	*lion*	false	NULL
2	$\forall_{eats.lion}$	true	1

s_sequences

s_id	i_id	s_depth
1	1	1

sl_sequences_levels

s_id	l_id
1	1

r_roles

r_id	r_uri
1	*eats*

l_levels

l_id	l_level	r_id
1	1	1

4 Performance Evaluation

Finally, there remains to explore the performance of the implementation described in Chapter 3 empirically. As mentioned in the introduction, a high load time is to be expected as a result of the fact, that the completion algorithm computes all consequences during the loading step. On the other hand, the query answering time should be low because we do not have to use a reasoner for answering queries and the backend database is designed to handle a huge amount of data and complex queries.

Section 4.1 will first introduce the benchmark setup and give a short description of other software which is used for the performance test of REAL. Afterwards, the different performance metrics are defined (Section 4.2) and the benchmark results are discussed (Section 4.3).

4.1 Benchmark Setup

First of all, the benchmark setup is heavily based on the paper "LUBM: A Benchmark for OWL Knowledge Base Systems"[7], which compares the load and query performance of selected OWL knowledge base systems. LUBM denotes "Lehigh University Benchmark". It is a benchmark system which contains an ontology with a data generator (UBA), a performance test application (UBT) and test queries. Furthermore, we use HAWK in order to compare the completion based repository REAL with another OWL knowledge base system. Altogether, the setup of the benchmark is illustrated in Figure 4.1. The next section describes the mentioned components and software briefly.

4.1.1 Software

Ontology (Univ-Bench)

The ontology used for testing is called "Univ-Bench"and models the domain of universities. It contains 43 classes (Department, Student, Professor, Course etc.) and 42 properties

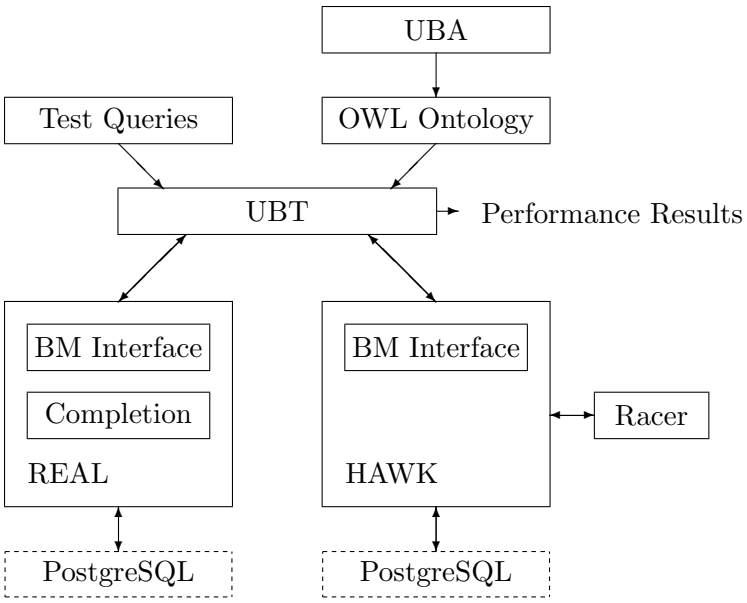

Figure 4.1: Benchmark setup.

(memberOf, degreeFrom, takesCourse etc.) which describe miscellaneous facets of an university. Moreover, there are different styles of defintions used. For example, the class `GraduateStudent` is built upon a restriction, whereas the class `UndergraduateStudent` is only defined as a subclass of the class `Student`.

The ontology is available as an OWL Lite ontology[1] (Version 1.0) and the description logic language defined in Section 2.2 is able to formalise it, although Univ-Bench misses a `owl:allValuesFrom` restriction which would be translated to an axiom of the form $\forall R.C$. Nevertheless, it is still possible to test the performance of REAL. See Appendix B.1 for a full listing of the OWL file. Note that the ontology does not include individuals or assertions about them and, speaking in terms of description logics, it represents only the TBox.

Data Generator (UBA)

The ABox assertions are created by a data generation application named UBA[2] (Version 1.7) which denotes "Univ-Bench Artificial data generator".

UBA allows users to repeatably generate different amounts of data. The smallest possible

[1] `http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl`
[2] `http://swat.cse.lehigh.edu/projects/lubm/`

size is one university which contains a varying number of departments[3], at least 15 and at most 25. Furthermore, each department contains 7 to 10 full professors, each one being the author of 15 to 20 publications and so on.[4] UBA creates a separate OWL file for each generated department. An abridgement of a data file is given in Appendix B.2.

Thus, to test REAL under different conditions, four datasets, called 01, 05, 10 and 20, are generated by UBA containing 1, 5, 10 and 20 universities respectively. The smallest dataset (01) contains about 100'000 triples[5] which equals a file size of 8 MB whereas the largest dataset (20) counts more than 2'700'000 triples with a total size of 234 MB. The benchmark made in the mentioned paper [7] uses 50 universities as the largest dataset. However, this huge amount of data (583 MB) can not be handled by REAL because the completion only uses the random accessed memory (RAM) and would, on the employed hardware, inevitably result in a memory overflow (see Section 4.1.2).

In order to run on UNIX based operating systems, the original UBA application, written in Java, had to be slightly modified. The modified source code can be found on the attached CD (see Appendix A.1).

Test Queries

LUBM defines 14 test queries in order to measure the performance of the knowledge base systems. Appendix B.3.1 describes the queries and lists them in the query language SPARQL, which is available as a working draft from W3C [11]. Furthermore, the tester UBT requires a configuration file for each tested repository, containing the used queries in a language the repository understands. In case of REAL the language is SQL and they are listed in Appendix B.3.2.

The queries are all of different complexity, depending on the number of involved concepts or role assertions, used class or role hierarchy and whether logical inference is required or not. The reader is referred to the document found in the bibliography as [7], which classifies and describes the 14 queries in detail.

[3] The number of departments is controlled by a seed value which is set to 0 (for at least 15 and at most 25 departments) for all datasets for the benchmark.

[4] A detailed description of the data profile can be found on the LUBM website `http://swat.cse.lehigh.edu/projects/lubm/profile.htm`.

[5] A subject, an object and a predicate are counted as one triple.

Tester (UBT)

The core of the LUBM package is the tester[6] (UBT) (Version 1.1). This application controls both data loading and query testing. The test schemes are fully customisable by using configuration files (attached on CD). Hence, the tester measures the loading and the query time together with the number of results and prints the data on screen. Nevertheless, to simplify the analysis of these results, the Java source code is modified in such a way, that UBT writes the results in a convenient format into files. The modified sources can be found on the attached CD (see Appendix A.1).

In order to test a knowledge base system with UBT, several interfaces of the UBT package have to be implemented in the repository. For REAL, this is done in the package `ubt` described in Section 3.1.1.

OWL Repository and Toolkit (HAWK)

HAWK[7] (Version 1.3 Beta) is a repository framework and toolkit that supports OWL. The application was developed, like LUBM, at Lehigh University and is the successor project of DLDB described in [10]. Moreover, HAWK supports memory and database based models which are, together with PostgreSQL DBMS, used for our benchmark test. However, even if we use the same DBMS as for REAL, the database scheme differs completely. HAWK creates for each class and for each property a table, which contains all individuals belonging to the class or all individuals that are related to the property respectively. Additionally, database views are used to model class or property hierarchy. For more information about the database design of Hawk, read the technical paper [10].

Unlike REAL, HAWK uses an external reasoner application called RacerPro[8] (Version 1.9.0) which has to be installed separately. Like the completion approach of REAL, HAWK uses the reasoner during the loading step and not during query execution. Altogether, HAWK is used as a reference repository for REAL regarding the benchmark test.

Database (PostgreSQL)

As already described in the Section 3.1.3, we use PostgreSQL (Version 8.1.3) as DBMS. We create a separate database for each dataset and repository. Additionally, the default

[6] `http://swat.cse.lehigh.edu/projects/lubm/`
[7] `http://swat.cse.lehigh.edu/downloads/`
[8] `http://www.racer-systems.com/`

configuration of PostgreSQL is modified to minimise query response time. The modified parameters are:

`geqo_threshold = 100` (default `12`) Use genetic query optimization to plan queries with at least this many SQL `FROM` items involved.

`effective_cache_size = 131072` (default `1000` disk pages) Effective size of the disk cache available to a single index scan as an assumption for the query planner.

`shared_buffers = 16384` (default `1000`) The number of shared memory buffers used by the database server.

`work_mem = 65536` (default `1024` KB) The amount of memory to be used by internal sort operations and hash tables.

Of course, these modifications depend strongly on the used hardware, especially the amount of available memory and the disk type. Moreover, increasing the shared memory buffer attribute of PostgreSQL requires also a higher limit of available shared memory of the operating system. (see Section 4.1.2).

4.1.2 Hardware, Operating System and JVM

The environment used for the benchmark test is as follows:

Hardware: Desktop Computer, 3 GHz Pentium 4 CPU, 2 GB RAM, 200 GB EIDE disk (7200 U/min)

Operating System: Debian[9] Etch Testing Release, Linux Kernel Version 2.4.26, allowed shared memory size increased[10] to 132 MB

Java Virtual Machine: Java Runtime Environment, Standard Edition Version 1.4.2_05, maximal heap size set to 1'800 MB

Note the increased heap size of the JVM. Otherwise, REAL fails to load the largest dataset 20.

[9] `http://www.debian.org`
[10] Execute `echo "138461184" > /proc/sys/kernel/shmmax`

4.2 Performance Metrics

4.2.1 Load Time

The load time is measured by UBT, which loads all generated OWL files of a dataset incrementally. The measuring starts when the first data file is read and stops when all data is stored in the database. Therefore, this metric includes the time for parsing the OWL files, reasoning or completion and storing the information in the database.

4.2.2 Repository Size

The repository size stands for the physical disk usage of the database after loading the data into the database. Therefore, this also includes all indexes or information about created views. Hence, the size of the database depends strongly on the implementation of the DBMS. However, both tested knowledge base systems are using PostgreSQL and therefore can be compared.

PostgreSQL does not support direct access to the physical size of a database. Nevertheless, the used disk space can be calculated by executing a shell command[11].

In addition, unused data (such as deleted database entries which are still stored in PostgreSQL) is deleted by the command `vacuumdb`[12] before and after each database load procedure to obtain meaningful results.

4.2.3 Query Response Time

Query response time is measured by UBT as listed in Algorithm 4.1. Note that each query is executed ten times to account for caching. Moreover, each query result is traversed sequentially.

4.2.4 Query Completeness and Soundness

As described in the LUBM paper [7], the degree of completeness of a query answer is measured by the percentage of the returned entailed answers compared to the total possible

[11] Change to the data directory of PostgreSQL (i.e. `cd /var/lib/postgresql/8.1/main/base`) and execute the OS disk usage program `du -hs [0-9]*` as a privileged user. The output shows the total disk usage for each database OID (internal number scheme of PostgreSQL). To translate the OID into the database name, execute the PostgreSQL helper application `oid2name`

[12] `vacuumdb -azf`

```
for all repositories r do
  for all datasets d do
    open repository r
    for all queries q do
      for i = 0 to 10 do
        start timer t
        execute query q
        traverse trough query result sequentially
        stop timer t
      end for
      compute average response time of t
    end for
    close repository r
  end for
end for
```

Algorithm 4.1: Procedure of the query response time measurement.

answers. On the other hand, the degree of soundness is calculated as the percentage of the answers of a query that are actually entailed.

In order to calculate the degree of soundness and completeness, we use a reference result set for all queries and datasets. The LUBM website[13] provides all answers for the queries of the dataset 01. Concerning the datasets 05, 10 and 20, we obtain the number of total answers from [7].

As mentioned in Section 2.3.1, REAL does not support annotations. Although there are annotations used in the queries, we consider a result of REAL to be complete and sound, if the correct individuals are returned even without their annotations.

4.3 Results and Discussion

4.3.1 Data Loading

Figure 4.2 shows how the load time and the database size increase in relation to the number of triples stored in the database. The precise figures are listed in Table 4.1. The number of table rows in the database of REAL are graphically plotted in Figure 4.3.

One remarkable issue is, that the load time of REAL increases exponentially while the database size growth is linear. The exponential load time of REAL can be explained by the completion algorithm in Section 3.2.1, where the search domain of a rule grows with each new assertions added by other rules.

[13] `http://swat.cse.lehigh.edu/projects/lubm/answers.htm`

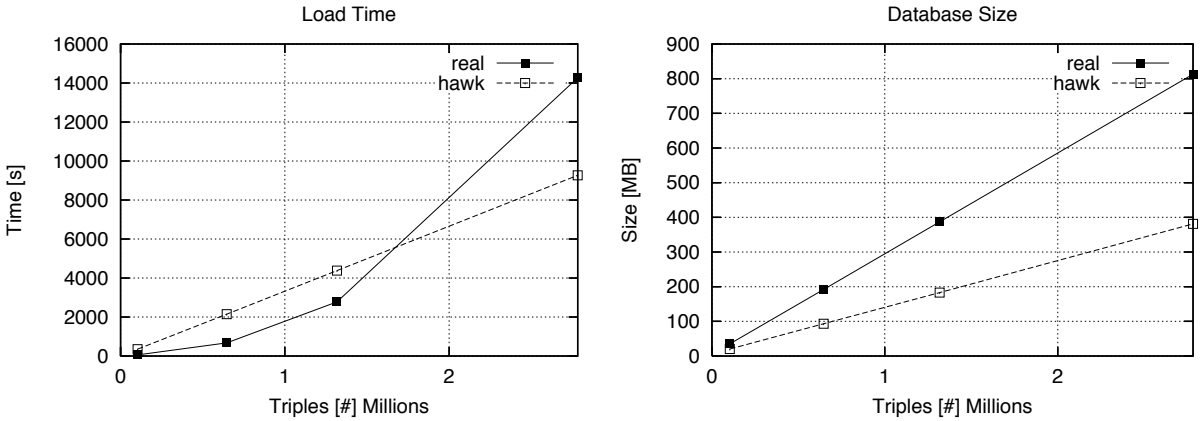

Figure 4.2: Load time compared to repository size.

On the other hand, the linear space consumption of REAL is not as expected in [12], which predicts exponential growth of the database instance. One possible reason for the linearity could be that the data generator produces datasets, for which the completion algorithm computes only a constant number of consequences for each new assertion included in the given dataset. Of course, this needs further investigation with other ontologies.

The load time and storage complexity of HAWK is linear. Concerning the database size, HAWK uses less storage than REAL. This had to be expected since HAWK does not add individuals to the tables in the case of concept or role inclusions. The hierarchy is mapped by database views which do not add additional data to the database.

	Dataset	**Size [MB]**	**Size [Triples]**	**Load Time [hh:mm:ss]**	**Database Size [MB]**
REAL	01	8,6	103'397	00:00:55	35
HAWK				00:05:53	20
REAL	05	54,2	646'128	00:11:17	192
HAWK				00:35:50	93
REAL	10	110,6	1'316'993	00:46:19	387
HAWK				01:13:00	183
REAL	20	234,8	2'782'419	03:57:49	813
HAWK				02:34:28	381

Table 4.1: Load time and repository size of REAL and HAWK.

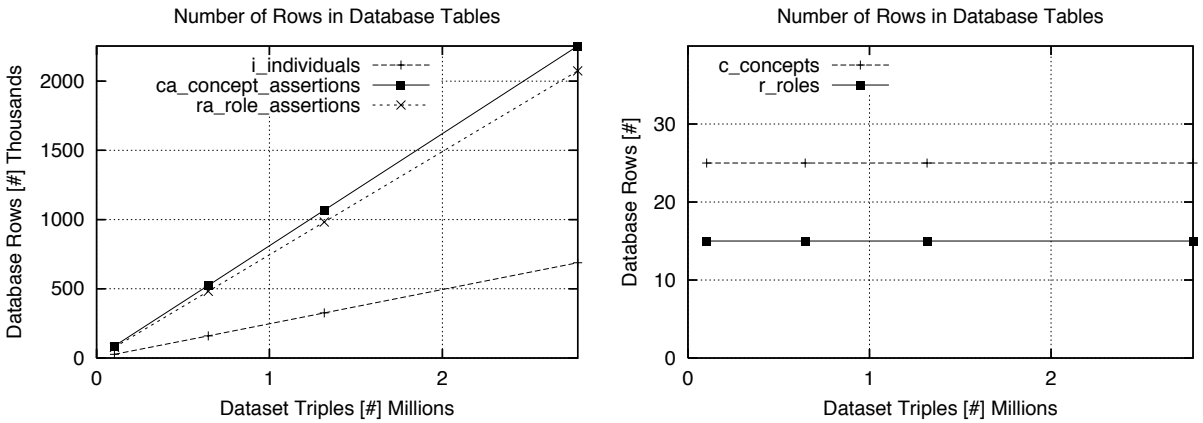

Figure 4.3: Number of rows in the relevant database tables of REAL compared to the number of loaded triples. The tables `s_sequences`, `sl_sequences_levels` and `l_levels` are not plotted because they do not contain any data.

4.3.2 Query Response Time

The query response time is given in three different forms. Figure 4.4 shows the comparison of the response time of each query with respect to each dataset. Figures 4.5, 4.6 and 4.7 depict the repository performance as the dataset size increases, with respect to each query. Finally, a complete list of the test results can be found in Appendix B.5.

At first glance, the absolute response time of the 14 queries in Figure 4.4 varies highly in all datasets. This is an indication that all the chosen queries differ in complexity and means that the database can not handle different queries equally. Obviously, the variation can not be explained by the fact that REAL uses another database design than HAWK since both repositories are affected.

If we look at each query's response time and its behaviour, we will notice that, for both repositories, the growth of the answering time needed for some queries can be classified as constant, linear or exponential. In the majority of cases (except queries 6 and 14), the type of growth differs between REAL and HAWK.

The answering time needed by the completion based repository can remains constant for queries 1, 3 to 5, 7, 10, 11 and 13. A common feature of these queries is that they all produce a constant number of results for each dataset. Nevertheless, if we look at queries 8 and 12, we cannot conclude that each query, which has a constant number of results for the datasets, can be answered in constant time. Therefore, there must be a difference in the type of those queries. All the former queries have in common that they contain one

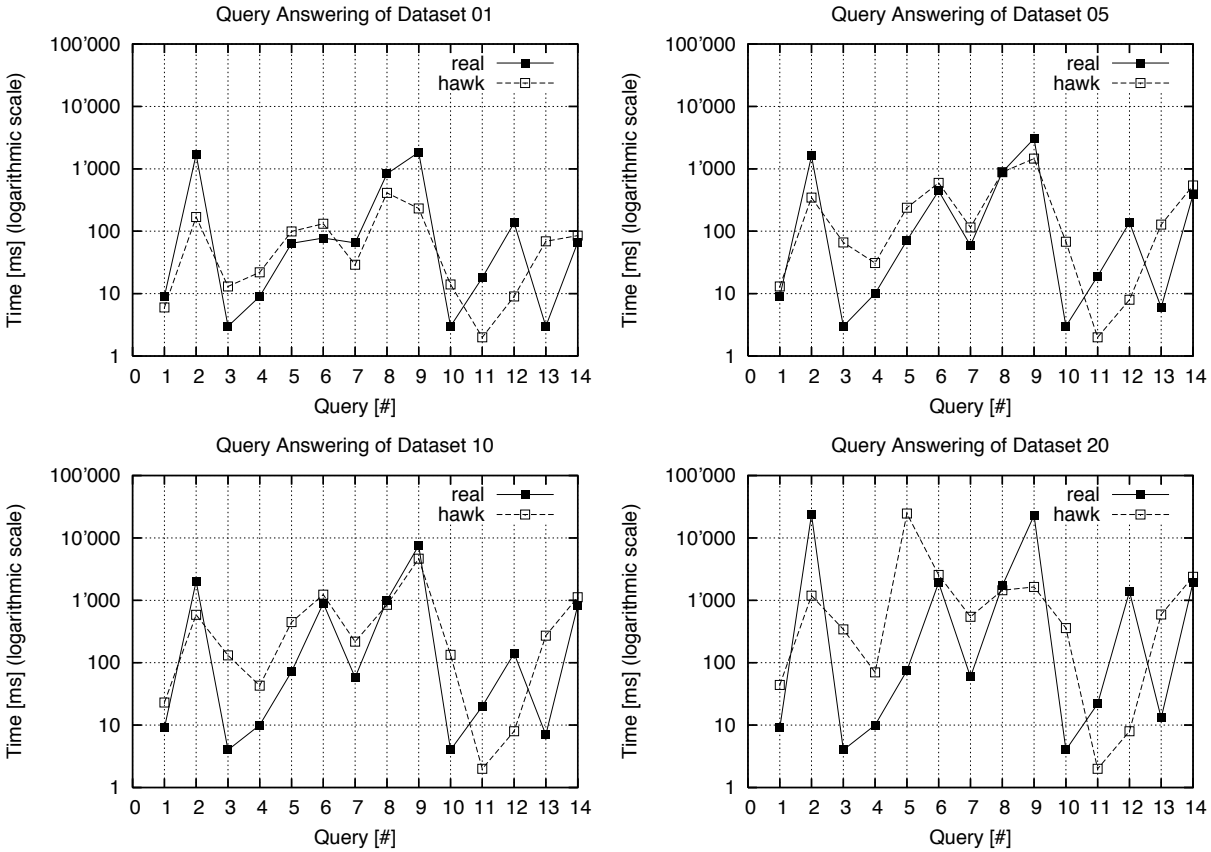

Figure 4.4: All datasets with the response time of each query. Note the logarithmic scale on the time axis.

or two concept assertions and one role assertion, where the role assertion always has a specific URI in the statement. Whereas queries 8 and 12, which are not always answered in the same time, contain two concept and two role assertions. Here, the role assertions are nested, because both share a variable and only one assertion uses a specific URI in the statement. The reason for this behaviour could be the additional join between the two role assertion tables. Of course, how these joins are finally executed is decided by the database query analyser.

Remarkable are also queries 2 and 9. Compared to the other queries, REAL needs a good amount of time even for the smallest test database. Both queries do not specify an URI but use a triangular pattern for role assertions involved. Obviously, our database design does not avail this sort of queries because HAWK can answer these queries faster and in an amount of time of linear growth (at least query 2 which returns a complete result).

Concerning the other queries, HAWK performs totally different. The only queries which are answered in the same period of time are 11 and 12, but their result is far away from being complete and thus not meaningful. The incompleteness of HAWK makes a reliable analysis difficult and will therefore be omitted.

Database Parameters

As mentioned in Section 4.1.1, the database configuration has been slightly modified for the test to increase performance. These modifications were made by trial and error. After each parameter change, a full benchmark was run until an "optimal"query answering time was found. The result of these optimisations are shown in Figure 4.8, where the performance of the modified configuration of PostgreSQL is compared to the default configuration. Note that, as an exception, the parameter `geqo_threshold` of the default configuration was modified. In order to get reliable results, it had to be set to `100` instead of the default value `12`. Otherwise, the query optimiser would be unable to produce deterministic execution plans within the 10 runs of each query.

Nevertheless, the configuration optimisations do not have a great impact. They can even lead to a worse performance (for example query 9 with dataset 10). Of course, if other hardware is used (i.e. SCSI drives, more memory) configuration changes are required in order to use the full potential of PostgreSQL.

4.3.3 Query Completeness and Soundness

The completeness of the query answers from all datasets is shown in Figure 4.9 and in Appendix B.5. As you may notice, results for the dataset 05, 10 and 20 and for queries 11 through 13 are missing. The reason for this absence is the lack of completeness results in reference papers or on the LUBM website. Nevertheless, we can see that in all other cases, REAL is able to give complete answers.

As for HAWK, which uses RacerPro as reasoner, it is in most of the cases incomplete. Dataset 01 returns the best result where only the answers of queries 11 and 12 were incomplete. These queries assume transitive role inference. The exact reason for this behaviour and why the other datasets produce more incomplete answers can not be given because this would require profound knowledge of Racer and HAWK.

There is no plot for the soundness but as far as dataset 01 is concerned, both repositories are sound. The other datasets were not tested because no reference answers exist.

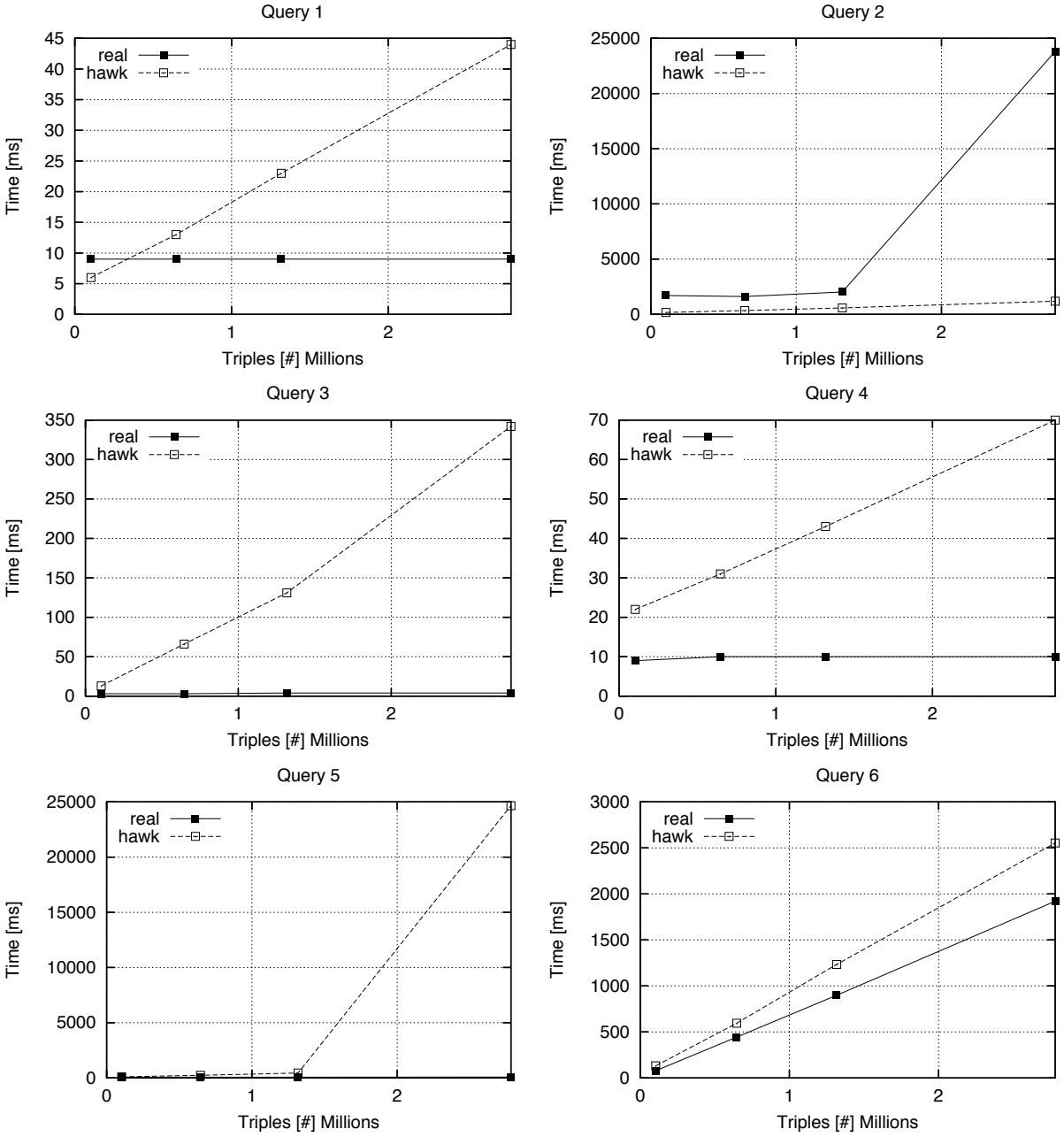

Figure 4.5: Response time of queries 1 to 6 in relation to the number of triples in the database.

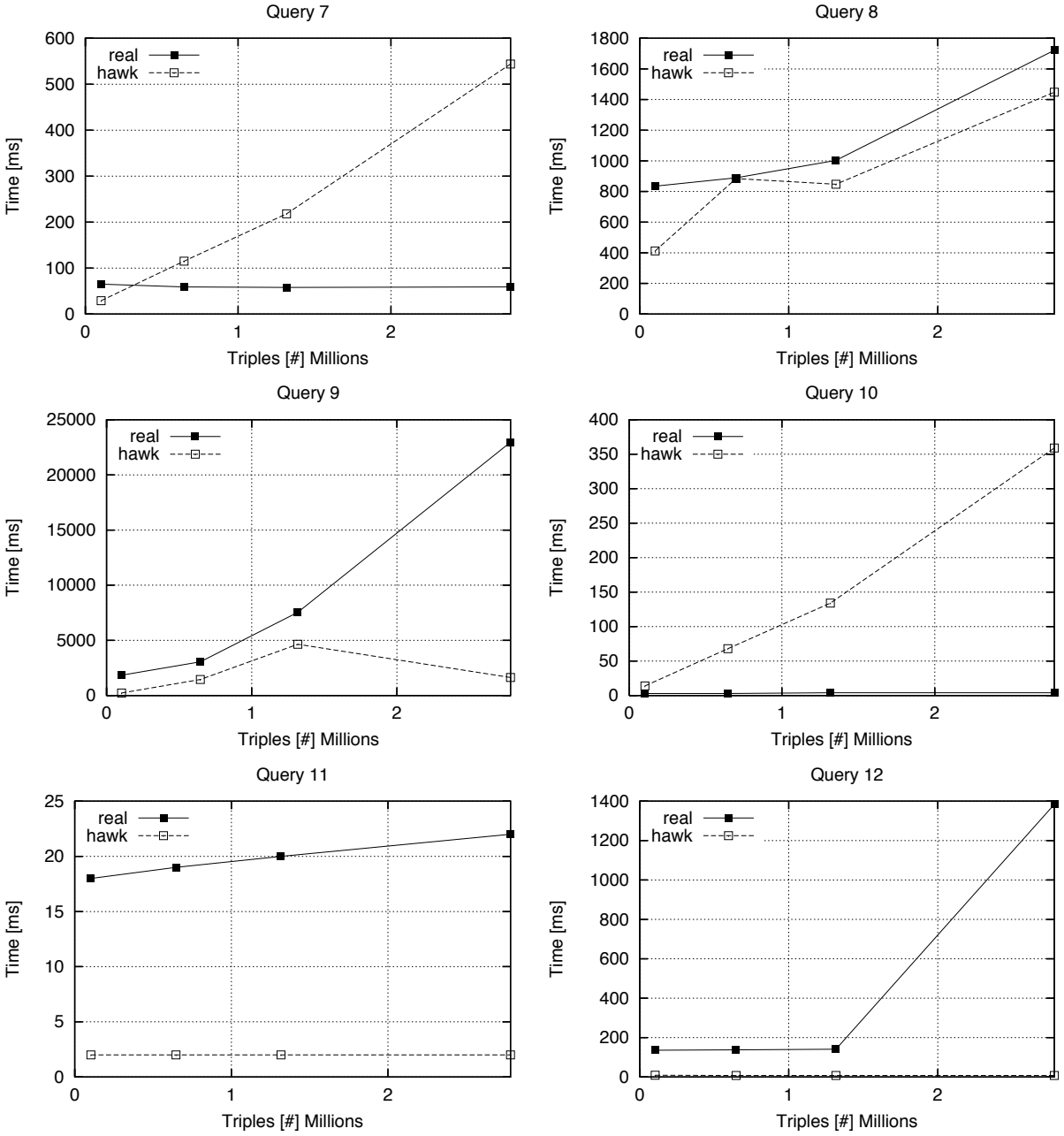

Figure 4.6: Response time of queries 7 to 12 in relation to the number of triples in the database.

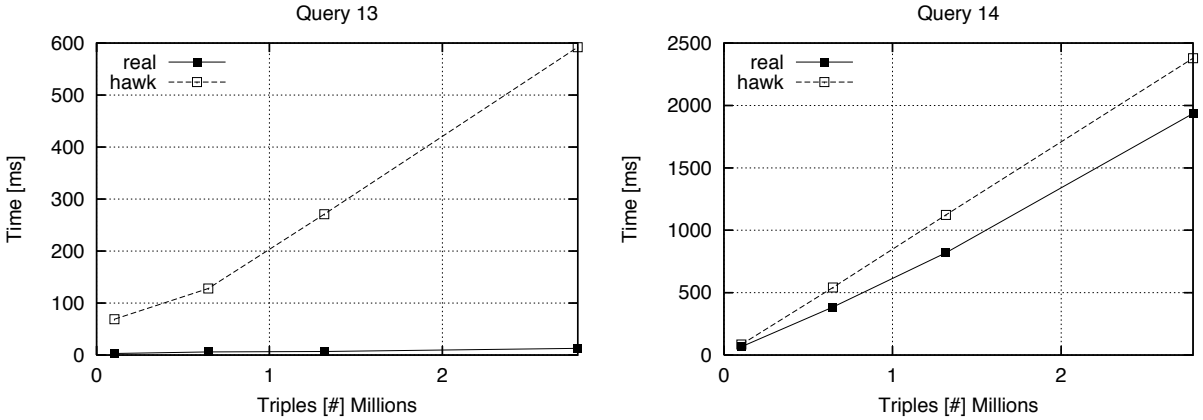

Figure 4.7: Response time of queries 13 and 14 in relation to the number of triples in the database.

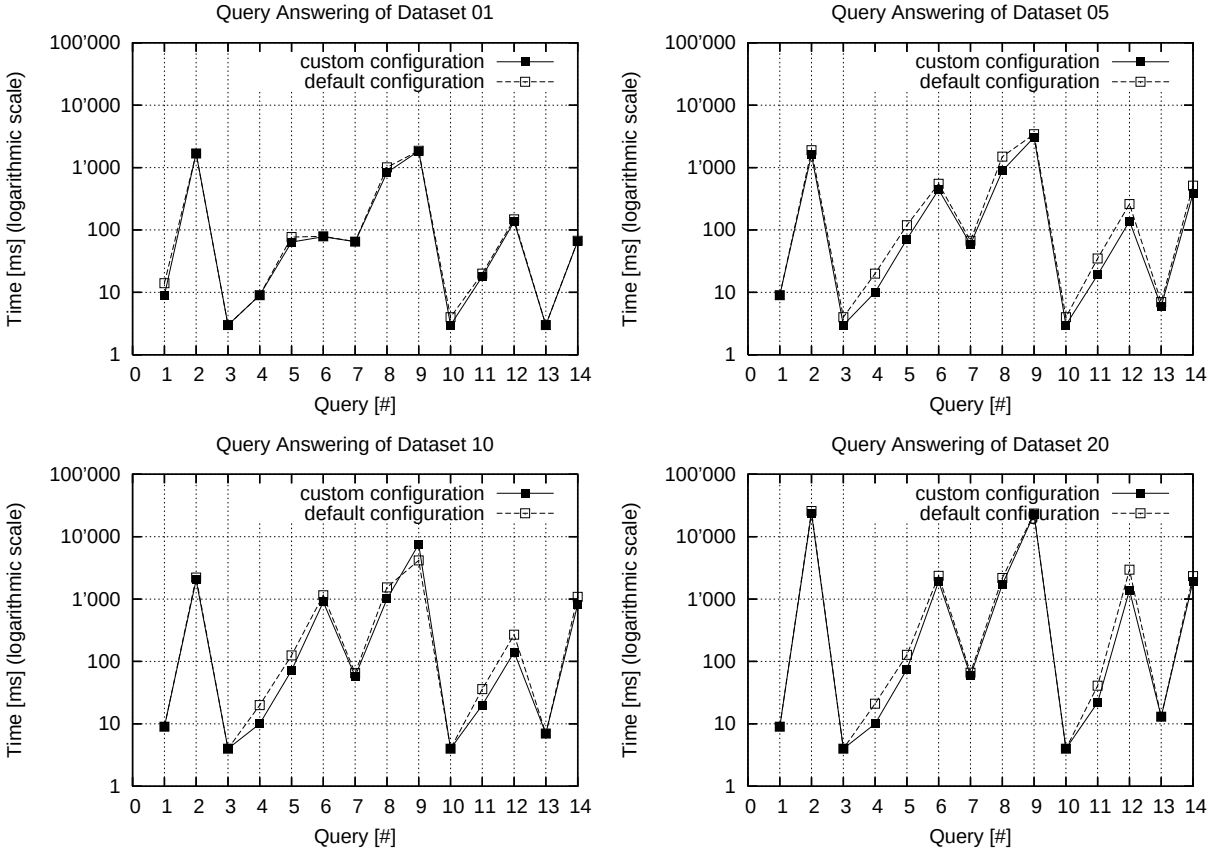

Figure 4.8: Comparison of query answering time between PostgreSQL default and optimised configuration as proposed in Section 4.1.1.

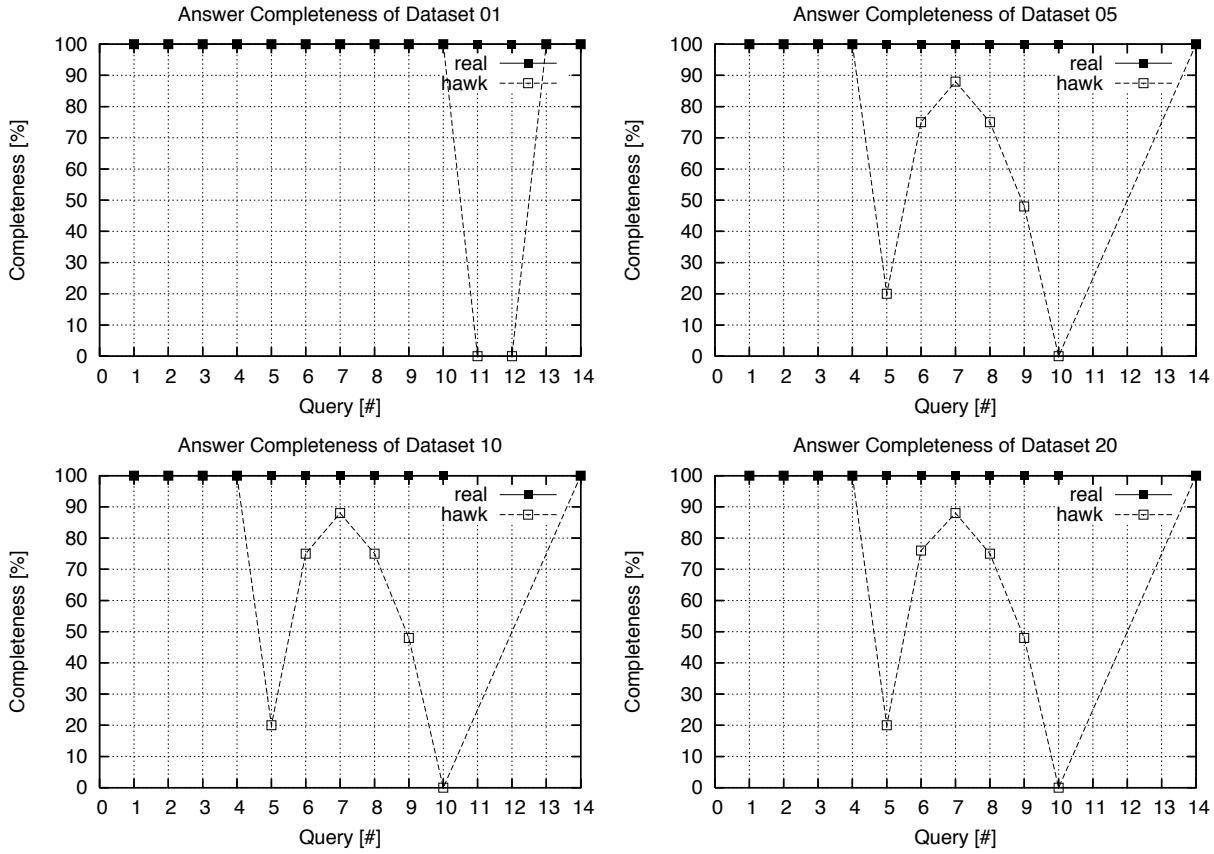

Figure 4.9: Query completeness.

5 Conclusion and Further Work

5.1 Conclusion

Supporting the claim made in the introduction, we showed that our knowledge base system, using a relational database, is able to provide sound and complete answers in reasonable time. Most of the queries are even answered in a constant amount of time depending on the repository size. This is remarkable as this behaviour is not obvious because the completion algorithm adds a lot of new individuals and assertions to the database. Of course, we have seen that the performance of the knowledge base system also depends on the database design used as well as on the complexity of the query and the number of returned results.

Furthermore, we have also realised that database configuration optimisation can improve performance, but their effect is minimal. Although the query planner of the database influences the performance the most. Nevertheless, a relational database system is a good choice for storing a knowledge base because it can handle huge amount of data in reasonable time.

Another result of the practical experiment is that the load time increases exponentially with respect to the size of the input. This behaviour is as expected whereas the linear growth of the database is not. This has to be further analysed with other test ontologies.

5.2 Further Work

The presented implementation of the knowledge base as well as the benchmark results lead to questions and proposals for further work:

- The linear storage complexity of the database is not as expected. This can be further analysed and verified with other test ontologies.
- The Univ-Bench test ontology, the data generator and the test queries could be modified to support value restrictions in order to test the performance when the

introduced special individual $\forall_{R.a}$ and the corresponding database tables come into play.

- REAL supports only memory based completion which limits the processing of very large datasets. The completion algorithm could be rewritten to use the database as temporal storage.
- Additionally, REAL could be enlarged to a stand-alone knowledge base application. Therefore, a query interface could be added which supports for example the language SPARQL.
- Finally, the implementation could be modified to support RDBMS other than PostgreSQL. A performance comparison would show if the benchmark results depend on the database management system.

A Implementation

A.1 Directory Tree

```
real/                 REAL REPOSITORY
    INSTALL           installation instructions
    real.jar          binary jar file
    doc/              documentation
        javadoc/      API
        thesis/       this thesis
    etc/              log configuration file
    lib/              external libraries
    log/              debug logs
    ontologies/       ontologies for unit tests
    scripts/          database initialisation script
    src/              project source files

REALBench/            BENCHMARK SUITE
    INSTALL           installation and benchmark instructions
    data/             test datasets created by UBA
    etc/              config files for repositories
    gnuplot/          gnuplot scripts and data
        custom/       custom draw scripts
        data/         temporal tables for gnuplot
        plots/        gnuplot command files
        postscripts/  final graphs as ps files
        scripts/      gnuplot draw scripts
        styles/       style files for graphs
        tmp/
```

```
    lib/            UBA, UBT, real and hawk libraries
    log/            log files
    query/          test queries for repositories
    scripts/        database size calculation scripts
    tests/          benchmark results

UBT/                TESTING APPLICATION
    readme.txt
    ubt.jar         binary jar file
    src/            project source files

UBA/                DATA GENERATOR
    readme.txt
    uba.jar         binary jar file
    src/            project source files

hawk/               HAWK REPOSITORY
    readme.txt
    hawk.jar        binary jar file
    lib/            external libraries
    src/            project source files
```

B Benchmark

B.1 Univ-Bench Ontology

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
   xmlns = "http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#"
   xml:base = "http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl"
   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
>

<owl:Ontology rdf:about="">
   <rdfs:comment>An university ontology for benchmark tests</rdfs:comment>
   <rdfs:label>Univ-bench Ontology</rdfs:label>
   <owl:versionInfo>univ-bench-ontology-owl, ver April 1, 2004</owl:versionInfo>
</owl:Ontology>

<owl:Class rdf:ID="AdministrativeStaff">
   <rdfs:label>administrative staff worker</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Employee" />
</owl:Class>

<owl:Class rdf:ID="Article">
   <rdfs:label>article</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Publication" />
</owl:Class>

<owl:Class rdf:ID="AssistantProfessor">
   <rdfs:label>assistant professor</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="AssociateProfessor">
   <rdfs:label>associate professor</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="Book">
   <rdfs:label>book</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Publication" />
```

```
</owl:Class>

<owl:Class rdf:ID="Chair">
  <rdfs:label>chair</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Department" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
  <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="ClericalStaff">
  <rdfs:label>clerical staff worker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#AdministrativeStaff" />
</owl:Class>

<owl:Class rdf:ID="College">
  <rdfs:label>school</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
</owl:Class>

<owl:Class rdf:ID="ConferencePaper">
  <rdfs:label>conference paper</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article" />
</owl:Class>

<owl:Class rdf:ID="Course">
  <rdfs:label>teaching course</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Work" />
</owl:Class>

<owl:Class rdf:ID="Dean">
  <rdfs:label>dean</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#College" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
  <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="Department">
  <rdfs:label>university department</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
```

```
</owl:Class>

<owl:Class rdf:ID="Director">
  <rdfs:label>director</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
  <owl:Restriction>
  <owl:onProperty rdf:resource="#headOf" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Program" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="Employee">
  <rdfs:label>Employee</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
  <owl:Restriction>
  <owl:onProperty rdf:resource="#worksFor" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Organization" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="Faculty">
  <rdfs:label>faculty member</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Employee" />
</owl:Class>

<owl:Class rdf:ID="FullProfessor">
  <rdfs:label>full professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="GraduateCourse">
  <rdfs:label>Graduate Level Courses</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Course" />
</owl:Class>

<owl:Class rdf:ID="GraduateStudent">
  <rdfs:label>graduate student</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Person" />
  <rdfs:subClassOf>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#takesCourse" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#GraduateCourse" />
  </owl:someValuesFrom>
```

```
  </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Institute">
  <rdfs:label>institute</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
</owl:Class>

<owl:Class rdf:ID="JournalArticle">
  <rdfs:label>journal article</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article" />
</owl:Class>

<owl:Class rdf:ID="Lecturer">
  <rdfs:label>lecturer</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty" />
</owl:Class>

<owl:Class rdf:ID="Manual">
  <rdfs:label>manual</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication" />
</owl:Class>

<owl:Class rdf:ID="Organization">
  <rdfs:label>organization</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Person">
  <rdfs:label>person</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PostDoc">
  <rdfs:label>post doctorate</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty" />
</owl:Class>

<owl:Class rdf:ID="Professor">
  <rdfs:label>professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Faculty" />
</owl:Class>

<owl:Class rdf:ID="Program">
  <rdfs:label>program</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
</owl:Class>

<owl:Class rdf:ID="Publication">
  <rdfs:label>publication</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Research">
```

```
  <rdfs:label>research work</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Work" />
</owl:Class>

<owl:Class rdf:ID="ResearchAssistant">
  <rdfs:label>university research assistant</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Person" />
  <rdfs:subClassOf>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#worksFor" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#ResearchGroup" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ResearchGroup">
  <rdfs:label>research group</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
</owl:Class>

<owl:Class rdf:ID="Schedule">
  <rdfs:label>schedule</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Software">
  <rdfs:label>software program</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication" />
</owl:Class>

<owl:Class rdf:ID="Specification">
  <rdfs:label>published specification</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication" />
</owl:Class>

<owl:Class rdf:ID="Student">
  <rdfs:label>student</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
  <owl:Restriction>
  <owl:onProperty rdf:resource="#takesCourse" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Course" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="SystemsStaff">
  <rdfs:label>systems staff worker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#AdministrativeStaff" />
```

```
</owl:Class>

<owl:Class rdf:ID="TeachingAssistant">
  <rdfs:label>university teaching assistant</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Person" />
  <owl:Restriction>
  <owl:onProperty rdf:resource="#teachingAssistantOf" />
  <owl:someValuesFrom>
  <owl:Class rdf:about="#Course" />
  </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="TechnicalReport">
  <rdfs:label>technical report</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Article" />
</owl:Class>

<owl:Class rdf:ID="UndergraduateStudent">
  <rdfs:label>undergraduate student</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Student" />
</owl:Class>

<owl:Class rdf:ID="University">
  <rdfs:label>university</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Organization" />
</owl:Class>

<owl:Class rdf:ID="UnofficialPublication">
  <rdfs:label>unnoficial publication</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Publication" />
</owl:Class>

<owl:Class rdf:ID="VisitingProfessor">
  <rdfs:label>visiting professor</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Professor" />
</owl:Class>

<owl:Class rdf:ID="Work">
  <rdfs:label>Work</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:ID="advisor">
  <rdfs:label>is being advised by</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#Professor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="affiliatedOrganizationOf">
  <rdfs:label>is affiliated with</rdfs:label>
```

```
  <rdfs:domain rdf:resource="#Organization" />
  <rdfs:range rdf:resource="#Organization" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="affiliateOf">
  <rdfs:label>is affiliated with</rdfs:label>
  <rdfs:domain rdf:resource="#Organization" />
  <rdfs:range rdf:resource="#Person" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="age">
  <rdfs:label>is age</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="degreeFrom">
  <rdfs:label>has a degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#University" />
  <owl:inverseOf rdf:resource="#hasAlumnus"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="doctoralDegreeFrom">
  <rdfs:label>has a doctoral degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#University" />
  <rdfs:subPropertyOf rdf:resource="#degreeFrom" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="emailAddress">
  <rdfs:label>can be reached at</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasAlumnus">
  <rdfs:label>has as an alumnus</rdfs:label>
  <rdfs:domain rdf:resource="#University" />
  <rdfs:range rdf:resource="#Person" />
  <owl:inverseOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="headOf">
  <rdfs:label>is the head of</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#worksFor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="listedCourse">
  <rdfs:label>lists as a course</rdfs:label>
  <rdfs:domain rdf:resource="#Schedule" />
  <rdfs:range rdf:resource="#Course" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="mastersDegreeFrom">
  <rdfs:label>has a masters degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#University" />
  <rdfs:subPropertyOf rdf:resource="#degreeFrom"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="member">
  <rdfs:label>has as a member</rdfs:label>
  <rdfs:domain rdf:resource="#Organization" />
  <rdfs:range rdf:resource="#Person" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="memberOf">
<rdfs:label>member of</rdfs:label>
<owl:inverseOf rdf:resource="#member" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="name">
<rdfs:label>name</rdfs:label>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="officeNumber">
  <rdfs:label>office room No.</rdfs:label>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="orgPublication">
  <rdfs:label>publishes</rdfs:label>
  <rdfs:domain rdf:resource="#Organization" />
  <rdfs:range rdf:resource="#Publication" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationAuthor">
  <rdfs:label>was written by</rdfs:label>
  <rdfs:domain rdf:resource="#Publication" />
  <rdfs:range rdf:resource="#Person" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationDate">
  <rdfs:label>was written on</rdfs:label>
  <rdfs:domain rdf:resource="#Publication" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publicationResearch">
  <rdfs:label>is about</rdfs:label>
  <rdfs:domain rdf:resource="#Publication" />
  <rdfs:range rdf:resource="#Research" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="researchInterest">
  <rdfs:label>is researching</rdfs:label>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:ID="researchProject">
  <rdfs:label>has as a research project</rdfs:label>
  <rdfs:domain rdf:resource="#ResearchGroup" />
  <rdfs:range rdf:resource="#Research" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="softwareDocumentation">
  <rdfs:label>is documented in</rdfs:label>
  <rdfs:domain rdf:resource="#Software" />
  <rdfs:range rdf:resource="#Publication" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="softwareVersion">
  <rdfs:label>is version</rdfs:label>
  <rdfs:domain rdf:resource="#Software" />
</owl:ObjectProperty>

<owl:TransitiveProperty rdf:ID="subOrganizationOf">
  <rdfs:label>is part of</rdfs:label>
  <rdfs:domain rdf:resource="#Organization" />
  <rdfs:range rdf:resource="#Organization" />
</owl:TransitiveProperty>

<owl:ObjectProperty rdf:ID="takesCourse">
  <rdfs:label>is taking</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="teacherOf">
  <rdfs:label>teaches</rdfs:label>
  <rdfs:domain rdf:resource="#Faculty" />
  <rdfs:range rdf:resource="#Course" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="teachingAssistantOf">
  <rdfs:label>is a teaching assistant for</rdfs:label>
  <rdfs:domain rdf:resource="#TeachingAssistant" />
  <rdfs:range rdf:resource="#Course" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="telephone">
  <rdfs:label>telephone number</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="tenured">
  <rdfs:label>is tenured:</rdfs:label>
  <rdfs:domain rdf:resource="#Professor" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="title">
  <rdfs:label>title</rdfs:label>
```

```
  <rdfs:domain rdf:resource="#Person" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="undergraduateDegreeFrom">
  <rdfs:label>has an undergraduate degree from</rdfs:label>
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#University" />
  <rdfs:subPropertyOf rdf:resource="#degreeFrom" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="worksFor">
  <rdfs:label>Works For</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#memberOf" />
</owl:ObjectProperty>

</rdf:RDF>
```

B.2 Example Dataset of UBA

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xml:base="http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl"
xmlns:ub="http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#">

<owl:Ontology rdf:about="http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl">
<owl:imports rdf:resource="http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl" />
</owl:Ontology>

<ub:University rdf:about="http://www.University0.edu">
   <ub:name>University0</ub:name>
</ub:University>

<ub:Department rdf:about="http://www.Department0.University0.edu">
   <ub:name>Department0</ub:name>
   <ub:subOrganizationOf>
      <ub:University rdf:about="http://www.University0.edu" />
   </ub:subOrganizationOf>
</ub:Department>

<ub:UndergraduateStudent rdf:about="http://www.Department0.University0.edu/
    UndergraduateStudent0">
   <ub:name>UndergraduateStudent0</ub:name>
   <ub:memberOf rdf:resource="http://www.Department0.University0.edu" />
   <ub:emailAddress>UndergraduateStudent0@Department0.University0.edu</ub:emailAddress>
   <ub:telephone>xxx-xxx-xxxx</ub:telephone>
   <ub:takesCourse rdf:resource="http://www.Department0.University0.edu/Course49" />
   <ub:takesCourse rdf:resource="http://www.Department0.University0.edu/Course3" />
   <ub:takesCourse rdf:resource="http://www.Department0.University0.edu/Course42" />
```

```
    <ub:takesCourse rdf:resource="http://www.Department0.University0.edu/Course4" />
</ub:UndergraduateStudent>

</rdf:RDF>
```

B.3 Test Queries of UBT

B.3.1 SPARQL Format

```
Query 1
# All the graduate students who take course
# http://www.Department0.University0.edu/GraduateCourse0
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:GraduateStudent .
       ?X ub:takesCourse http://www.Department0.University0.edu/GraduateCourse0}

Query 2
# All the graduate students who are now studying at the university from
# which they obtained their bachelor degrees
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE {?X rdf:type ub:GraduateStudent .
       ?Y rdf:type ub:University .
       ?Z rdf:type ub:Department .
       ?X ub:memberOf ?Z .
       ?Z ub:subOrganizationOf ?Y .
       ?X ub:undergraduateDegreeFrom ?Y}

Query 3
# All the publications of professor
# http://www.Department0.University0.edu/AssistantProfessor0
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:Publication .
       ?X ub:publicationAuthor http://www.Department0.University0.edu/AssistantProfessor0}

Query 4
# All the professors at Department0 of University0 and their email addresses
# and telephone numbers
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3
WHERE {?X rdf:type ub:Professor .
       ?X ub:worksFor <http://www.Department0.University0.edu> .
       ?X ub:name ?Y1 .
       ?X ub:emailAddress ?Y2 .
       ?X ub:telephone ?Y3}
```

```
Query 5
# All the members of department http://www.Department0.University0.edu
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:Person .
       ?X ub:memberOf <http://www.Department0.University0.edu>}

Query 6
# All the students
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:Student}

Query 7
# All the students who take courses of professor
# http://www.Department0.University0.edu/AssociateProfessor0
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y
WHERE {?X rdf:type ub:Student .
       ?Y rdf:type ub:Course .
       ?X ub:takesCourse ?Y .
       <http://www.Department0.University0.edu/AssociateProfessor0>, ub:teacherOf, ?Y}

Query 8
# All the students of university
# http://www.University0.edu and their email addresses
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE {?X rdf:type ub:Student .
       ?Y rdf:type ub:Department .
       ?X ub:memberOf ?Y .
       ?Y ub:subOrganizationOf <http://www.University0.edu> .
       ?X ub:emailAddress ?Z}

Query 9
# All the students who take the courses of their advisors.
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE {?X rdf:type ub:Student .
       ?Y rdf:type ub:Faculty .
       ?Z rdf:type ub:Course .
       ?X ub:advisor ?Y .
       ?Y ub:teacherOf ?Z .
       ?X ub:takesCourse ?Z}

Query 10
```

```
# All the students who take course
# http://www.Department0.University0.edu/GraduateCourse0
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:Student .
       ?X ub:takesCourse <http://www.Department0.University0.edu/GraduateCourse0>}

Query 11
# All the research groups at university http://www.University0.edu
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:ResearchGroup .
       ?X ub:subOrganizationOf <http://www.University0.edu>}

Query 12
# All the department chairs of university http://www.University0.edu
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y
WHERE {?X rdf:type ub:Chair .
       ?Y rdf:type ub:Department .
       ?X ub:worksFor ?Y .
       ?Y ub:subOrganizationOf <http://www.University0.edu>}

Query 13
# All the alumni of university http://www.University0.edu
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-
SELECT ?X
WHERE {?X rdf:type ub:Person .
       <http://www.University0.edu> ub:hasAlumnus ?X}


Query 14
# All the undergraduate students
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:UndergraduateStudent}
```

B.3.2 SQL Format for REAL

```
# Queries of UBT benchmark test for repository REAL

[01]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#GraduateStudent'
```

```
AND     ca1.i_id = x.i_id
AND     r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse'
AND     ra1.r_id = r1.r_id
AND     ra1.i_id_domain = x.i_id
AND     ra1.i_id_range = i1.i_id
AND     i1.i_uri = 'http://www.Department0.University0.edu/GraduateCourse0'

[02]
SELECT  x.i_uri
FROM    i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
        ca_concept_assertions AS ca2, c_concepts AS c2,
        ca_concept_assertions AS ca3, c_concepts AS c3,
        ra_role_assertions AS ra1, r_roles AS r1,
        ra_role_assertions AS ra2, r_roles AS r2,
        ra_role_assertions AS ra3, r_roles AS r3
WHERE   c1.c_id = ca1.c_id
AND     c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#GraduateStudent'
AND     ca1.i_id = x.i_id
AND     c2.c_id = ca2.c_id
AND     c2.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#University'
AND     c3.c_id = ca3.c_id
AND     c3.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Department'
AND     ra1.r_id = r1.r_id
AND     r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#memberOf'
AND     ra1.i_id_domain = x.i_id
AND     ra1.i_id_range = ca3.i_id
AND     ra2.r_id = r2.r_id
AND     r2.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    subOrganizationOf'
AND     ra2.i_id_domain = ca3.i_id
AND     ra2.i_id_range = ca2.i_id
AND     ra3.r_id = r3.r_id
AND     r3.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    undergraduateDegreeFrom'
AND     ra3.i_id_domain = x.i_id
AND     ra3.i_id_range = ca2.i_id

[03]
SELECT  x.i_uri
FROM    i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
        r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE   c1.c_id = ca1.c_id
AND     c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Publication'
AND     ca1.i_id = x.i_id
AND     r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    publicationAuthor'
AND     ra1.r_id = r1.r_id
AND     ra1.i_id_domain = x.i_id
AND     ra1.i_id_range = i1.i_id
AND     i1.i_uri = 'http://www.Department0.University0.edu/AssistantProfessor0'

[04]
```

```
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Professor'
AND      ca1.i_id = x.i_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#worksFor'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = i1.i_id
AND      i1.i_uri = 'http://www.Department0.University0.edu'

[05]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Person'
AND      ca1.i_id = x.i_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#memberOf'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = i1.i_id
AND      i1.i_uri = 'http://www.Department0.University0.edu'

[06]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student'
AND      ca1.i_id = x.i_id

[07]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         ca_concept_assertions AS ca2, c_concepts AS c2,
         ra_role_assertions AS ra1, r_roles AS r1,
         ra_role_assertions AS ra2, r_roles AS r2,
         i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student'
AND      ca1.i_id = x.i_id
AND      ca2.c_id = c2.c_id
AND      c2.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Course'
AND      ra1.r_id = r1.r_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse'
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = ca2.i_id
AND      ra2.r_id = r2.r_id
AND      r2.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf'
AND      ra2.i_id_domain = i1.i_id
AND      ra2.i_id_range = ca2.i_id
```

```
AND     i1.i_uri = 'http://www.Department0.University0.edu/AssociateProfessor0'

[08]
SELECT  x.i_uri
FROM    i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
        ca_concept_assertions AS ca2, c_concepts AS c2,
        ra_role_assertions AS ra1, r_roles AS r1,
        ra_role_assertions AS ra2, r_roles AS r2,
        i_individuals AS i1
WHERE   c1.c_id = ca1.c_id
AND     c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student'
AND     ca1.i_id = x.i_id AND
AND     c2.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Department'
AND     c2.c_id = ca2.c_id
AND     ra1.r_id = r1.r_id
AND     r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#memberOf'
AND     ra1.i_id_domain = x.i_id
AND     ra1.i_id_range = ca2.i_id
AND     ra2.r_id = r2.r_id
AND     r2.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    subOrganizationOf'
AND     ra2.i_id_domain = ca2.i_id
AND     ra2.i_id_range = i1.i_id
AND     i1.i_uri = 'http://www.University0.edu'

[09]
SELECT  x.i_uri
FROM    i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
        ca_concept_assertions AS ca2, c_concepts AS c2,
        ca_concept_assertions AS ca3, c_concepts AS c3,
        ra_role_assertions AS ra1, r_roles AS r1,
        ra_role_assertions AS ra2, r_roles AS r2,
        ra_role_assertions AS ra3, r_roles AS r3
WHERE   c1.c_id = ca1.c_id AND
AND     c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student'
AND     ca1.i_id = x.i_id
AND     c2.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Faculty'
AND     c2.c_id = ca2.c_id
AND     c3.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Course'
AND     c3.c_id = ca3.c_id
AND     ra1.r_id = r1.r_id
AND     r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor'
AND     ra1.i_id_domain = x.i_id
AND     ra1.i_id_range = ca2.i_id
AND     ra2.r_id = r2.r_id
AND     r2.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf'
AND     ra2.i_id_domain = ca2.i_id
AND     ra2.i_id_range = ca3.i_id
AND     ra3.r_id = r3.r_id
AND     r3.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse'
AND     ra3.i_id_domain = x.i_id
AND     ra3.i_id_range = ca3.i_id
```

```
[10]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student'
AND      ca1.i_id = x.i_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = i1.i_id
AND      i1.i_uri = 'http://www.Department0.University0.edu/GraduateCourse0'


[11]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#ResearchGroup'
AND      ca1.i_id = x.i_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    subOrganizationOf'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = i1.i_id
AND      i1.i_uri = 'http://www.University0.edu'

[12]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         c_concepts AS c2, ca_concept_assertions AS ca2,
         r_roles AS r1, ra_role_assertions AS ra1,
         r_roles AS r2, ra_role_assertions AS ra2,
         i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Chair'
AND      ca1.i_id = x.i_id
AND      c2.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Department'
AND      ca2.c_id = c2.c_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#worksFor'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = x.i_id
AND      ra1.i_id_range = ca2.i_id
AND      r2.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    subOrganizationOf'
AND      ra2.r_id = r2.r_id
AND      ra2.i_id_domain = ca2.i_id
AND      ra2.i_id_range = i1.i_id
AND      i1.i_uri = 'http://www.University0.edu'
```

```
[13]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1,
         r_roles AS r1, ra_role_assertions AS ra1, i_individuals AS i1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Person'
AND      ca1.i_id = x.i_id
AND      r1.r_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#hasAlumnus'
AND      ra1.r_id = r1.r_id
AND      ra1.i_id_domain = i1.i_id
AND      ra1.i_id_range = x.i_id
AND      i1.i_uri = 'http://www.University0.edu'

[14]
SELECT   x.i_uri
FROM     i_individuals AS x, c_concepts AS c1, ca_concept_assertions AS ca1
WHERE    c1.c_id = ca1.c_id
AND      c1.c_uri = 'http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#
    UndergraduateStudent'
AND      ca1.i_id = x.i_id
```

B.4 UBT Configuration Files

B.5 Test Results

	Dataset	01		05		10		20	
Query	Metric	REAL	HAWK	REAL	HAWK	REAL	HAWK	REAL	HAWK
1	Time [ms]	9	6	9	13	9	23	9	44
	Answers [#]	4	4	4	4	4	4	4	4
	Completeness [%]	100	100	100	100	100	100	100	100
2	Time [ms]	1700	169	1613	345	2025	587	23814	1188
	Answers [#]	0	0	9	9	28	28	59	59
	Completeness [%]	100	100	100	100	100	100	100	100
3	Time [ms]	3	13	3	66	4	131	4	342
	Answers [#]	6	6	6	6	6	6	6	6
	Completeness [%]	100	100	100	100	100	100	100	100
4	Time [ms]	9	22	10	31	10	43	10	70
	Answers [#]	34	34	34	34	34	34	34	34
	Completeness [%]	100	100	100	100	100	100	100	100
5	Time [ms]	64	99	71	238	72	447	75	24652
	Answers [#]	719	719	719	146	719	146	719	146
	Completeness [%]	100	100	100	20	100	20	100	20
6	Time [ms]	78	132	444	595	899	1234	1921	2553
	Answers [#]	7790	7790	48582	36682	99566	75547	210603	160120
	Completeness [%]	100	100	100	75	100	75	100	76
7	Time [ms]	65	29	59	115	58	218	59	544
	Answers [#]	67	67	67	59	67	59	67	59
	Completeness [%]	100	100	100	88	100	88	100	88
8	Time [ms]	835	412	889	884	1003	847	1722	1448
	Answers [#]	7790	7790	7790	5916	7790	5916	7790	5916
	Completeness [%]	100	100	100	75	100	75	100	75
9	Time [ms]	1844	232	3053	1458	7547	4651	22968	1636
	Answers [#]	208	208	1245	600	2540	1233	5479	2637
	Completeness [%]	100	100	100	48	100	48	100	48
10	Time [ms]	3	14	3	68	4	134	4	359
	Answers [#]	4	4	4	0	4	0	4	0
	Completeness [%]	100	100	100	0	100	0	100	0
11	Time [ms]	18	2	19	2	20	2	22	2
	Answers [#]	224	0	224	0	224	0	224	0
	Completeness [%]	100	0						
12	Time [ms]	136	9	138	8	141	8	1384	8
	Answers [#]	15	0	15	0	15	0	15	0
	Completeness [%]	100	0						
13	Time [ms]	3	69	6	128	7	271	13	592
	Answers [#]	1	1	21	14	33	18	86	50
	Completeness [%]	100	100						
14	Time [ms]	66	85	385	541	820	1123	1936	2379
	Answers [#]	5916	5916	36682	36682	75547	75547	160120	160120
	Completeness [%]	100	100	100	100	100	100	100	100

Bibliography

[1] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004. ISBN 0-262-01210-3.

[2] Franz Baader, Diego Calavanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003. ISBN 0-521-78176-0.

[3] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, L. McGuinness, Deborah, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

[4] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.1, November 2003. URL `http://www.w3.org/TR/2003/PR-xml-names11-20031105/`.

[5] Wikipedia The Free Encyclopedia. Knowledge Base, June 2005. URL `http://en.wikipedia.org/wiki/Knowledge_base`.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Massachusetts, 1994. ISBN 0-201-63361-2.

[7] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *J. Web Sem.*, 3(2-3):158–182, 2005. URL `http://www.websemanticsjournal.org/ps/pub/2005-16`.

[8] Jeff Heflin. OWL Web Ontology Language Use Cases and Requirements, February 2004. URL `http://www.w3.org/TR/2004/REC-webont-req-20040210/`.

[9] L. McGuinness, Deborah and Frank van Harmelen. OWL Web Ontology Language Overview, February 2004. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[10] Zhengxiang Pan and Jeff Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. Technical Report LU-CSE-04-006, Dept. of Computer Science and Engineering, Lehigh University, 2004. URL `http://swat.cse.lehigh.edu/pubs/pan04a.pdf`.

[11] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, October 2004. URL `http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/`.

[12] Thomas Studer. Relational Representation of ALN Knowledge Bases. In P. Isaias, M. Nunes, and A. Palma dos Reis, editors, *Proceedings of Multi*, pages 271–278. IADIS, 2005. URL `http://www.iam.unibe.ch/~tstuder/papers/mccsis.pdf`.