Querying the Semantic Web with Racer + nRQL

Volker Haarslev[1], Ralf Möller[2], and Michael Wessel[2]

[1] Concordia University, Montreal
Email: haarslev@cs.concordia.ca
[2] Technical University of Hamburg-Harburg
Email: {r.f.moeller | mi.wessel}@tuhh.de

Abstract. This paper introduces a description logic query language for retrieving A-box individuals that satisfy specific conditions. The language is substantially more expressive than traditional concept-based retrieval languages offered by previous description logic reasoning systems. The new language is implemented in the Racer system. We demonstrate the applicability of nRQL (new Racer Query Language) to OWL semantic web repositories and evaluate the performance of the current state of the art query answering engines for description logics using the *Lehigh University Benchmark (LUBM).*

1 Motivation

The semantic web is aimed at providing machine "understandable" meta data information for web resources. The most prominent semantic web meta data markup language is OWL/RDF. Its design is heavily influenced by description logics (DLs) [1]. Racer [2, 3] is a description logic system for the very expressive DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$. Racer can also process OWL Lite knowledge bases, as well as OWL DL with approximations for nominals, see also [4]. Racer's most prominent feature is that it is a true *A-box reasoner;* i.e., Racer not only provides means for representing meta data schema described in OWL ontologies, but also extensionally specified information, e.g., representing the actual web resources with concrete individuals and their interrelationships. Obviously, as such it is not only important to reason about web resources on an intensional level, but also being able to actually *extensionally query* these resources.

In this paper we show how Racer's new A-box query language nRQL(new Racer query language) can be used to provide access to extensionally specified information in A-boxes. We demonstrate the applicability of nRQL (pronounce: *nercle*) to OWL semantic web repositories and evaluate the performance of the current state of the art query answering engines for description logics using the *Lehigh University Benchmark (LUBM).*

The language nRQL augments and extends Racer's functional API for querying a knowledge base (a knowledge base, KB for short, is simply a T-box/A-box tuple $(\mathcal{T}, \mathcal{A})$. For instance, Racer provides a query function for retrieving all individuals mentioned in an A-box that are instances of a given query concept. Let us consider the A-box $\{has\_child(alice, betty),\ has\_child(alice, charles)\}$. If we are interested in finding individuals for which it can be proven that a child exists, in the Racer system, the function *concept_instances* can be used. However, if we would like to find all tuples of individuals x and y such that a common parent exists, currently, it is not possible to directly express this in sound and complete DL systems such as, for instance, Racer.

The paper is structured as follows: first we introduce the new Racer query language. Basic knowledge in description logics is necessary to understand the introduction to nRQL. We discuss and demonstrate the utility of nRQL with examples. We then demonstrate the feasibility of the approach by benchmarking the performance of Racer + nRQL using the so-called *Lehigh University Benchmark (LUBM)* [5]. Finally, we briefly discuss the relationship to OWL-QL [6] and comment on future research.

2 The New Racer Query Language - nRQL

Racer supports the DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ with, for instance, algebraic constraints (concrete domains) such as linear inequations for the reals and equations for strings. In the following we describe the syntax and semantics of nRQL. We start with some auxiliary definitions:

Definition 1 (Individuals, Variables, Object Names). *Let $\mathcal{I}$ and $\mathcal{V}$ be two disjoint sets of* individuals *and* variables*, respectively. The set $\mathcal{O} =_{def} \mathcal{V} \cup \mathcal{I}$ is the set of* object names*. We denote variables with letters $x, y, \ldots$; individuals are named $i, j, \ldots$; and object names with $a, b, \ldots$.*

Query atoms are the basic syntax expressions of nRQL:

Definition 2 (Query Atoms). *Let $a, b \in \mathcal{O}$; C be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ concept expression, R a nRQL role expression (a nRQL role expression is either a $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ role expression, or a* negated *$\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ role expression); P one of the concrete domain expressions offered by Racer; and f, g be so-called attributes (whose range is defined to be one of the available concrete domains offered by Racer). Then, the set of* nRQL atoms *is given as follows:*

- *Unary concept query atoms: $C(a)$*
- *Binary role query atoms: $R(a, b)$*
- *Binary constraint query atoms: $P(f(a), g(b))$*
- *Binary same-as atoms: $same\_as(a, i)$*
- *Unary has-known-successor atoms: $has\_known\_successor(a, R)$*
- *Negated atoms: If A is a nRQL atom, then so is $\backslash(A)$, a so-called* negation as failure atom *or simply* negated atom.

nRQL also offers various syntactic abbreviations ("macros") to facilitate query formulation. For example, it is possible to use *role chains followed by an attribute* within binary constraint query atoms: the atom $((\lambda(age_1, age_2).(age_1 + 70) < age_2)(age(has\_child(x)), age(has\_father(has\_father(x)))))$ retrieves all individuals x having a child being at least 70 years younger than its grand grandfather. This given atom is just an abbreviation for the complex conjunction $has\_child(x, y) \wedge$ $has\_father(x, z) \wedge$ $has\_father(z, u) \wedge$ $((\lambda(age_1, age_2).(age_1 + 70) < age_2)(age(y), age(u)))$.

Definition 3 (nRQL Queries). *A* nRQL Query *has a* head *and a* body. Query bodies *are defined inductively as follows:*

- *Each nRQL atom A is a body; and*
- *If $b_1 \ldots b_n$ are bodies, then the following are also bodies:*
  - $b_1 \wedge \cdots \wedge b_n$*,* $b_1 \vee \cdots \vee b_n$*,* $\backslash(b_i)$

We use the syntax $body(a_1, \ldots, a_n)$ to indicate that $a_1, \ldots, a_n$ are all the object names ($a_i \in \mathcal{O}$) mentioned in body. A nRQL Query *is then an expression of the form*

$$ans(a_{i_1}, \ldots, a_{i_m}) \leftarrow body(a_1, \ldots, a_n),$$

The expression $ans(a_{i_1}, \ldots, a_{i_m})$ is also called the head*, and $(i_1, \ldots, i_m)$ is an index vector with $i_j \in 1 \ldots n$. A* conjunctive nRQL query *is a query which does not contain any $\vee$ and $\backslash$ operators.*

The predicate *ans* should be understood as a keyword. Note that, similar to other logical languages, it is also possible to use individuals as "constants" in the head of queries.

Before we consider atoms with variables, we define truth of *ground query atoms.*

Definition 4 (Ground Query Atoms). *A ground query atom is a query atom that does not contain any variables. A positive ground query atom is a ground query atom that is not negated.*

To define truth of ground query atoms, we will need the standard notion of *logical implication* or *logical entailment*. Assuming that $\mathsf{inds}(\mathcal{A})$ returns the set of all individuals mentioned in A-box $\mathcal{A}$ we first start with *positive atoms*.

Definition 5 (Entailment of Positive Ground Query Atoms). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ knowledge base.*

A positive ground query atom A is logically entailed (or implied) by $\mathcal{K}$ iff every model $\mathcal{I}$ of $\mathcal{K}$ is also a model of A. In this case we write $\mathcal{K} \models A$. Moreover, if $\mathcal{I}$ is a model of $\mathcal{K}$ (A) we write $\mathcal{I} \models \mathcal{K}$ ($\mathcal{I} \models A$). We therefore have to specify when $\mathcal{I} \models A$ holds. In the following, if the atom A contains individuals i, j, it will always be the case that $i, j \in \mathsf{inds}(\mathcal{A})$. From this it follows that $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $j^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for any $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\mathcal{I} \models \mathcal{K}$:

- *If $A = C(i)$, then $\mathcal{I} \models A$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$.*
- *If $A = R(i, j)$, then $\mathcal{I} \models A$ iff $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$.*
- *If $A = P(f(i), g(j))$, then $\mathcal{I} \models A$ iff $(f^{\mathcal{I}}(i^{\mathcal{I}}), g^{\mathcal{I}}(j^{\mathcal{I}})) \in P^{\mathcal{I}}$.*
- *If $A = same\_as(i, i)$, then $\mathcal{I} \models A$.*
- *If $A = same\_as(i, j)$, then $\mathcal{I} \not\models A$.*
- *If $A = has\_known\_successor(i, R)$, then $\mathcal{I} \models A$ iff for some $j \in \mathsf{inds}(\mathcal{A})$: $\mathcal{I} \models R(i, j)$.*

It is important to note that the properties of roles and concepts referred to in the query atoms are defined in the knowledge base $\mathcal{K}$. For example, if the role *has_descendant* has been declared as transitive in $\mathcal{K}$, then *has_descendant* is transitive in the queries as well, since in models of $\mathcal{K}$ $\mathit{has\_descendant}^{\mathcal{I}} = (\mathit{has\_descendant}^{\mathcal{I}})^+$ must hold. If has_father is declared as a feature (a functional role), then it is treated as a feature in queries as well.

Also note that $\mathit{same\_as}(i, j)$ is basically a *syntactical* notion. Suppose that in some models $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $i^{\mathcal{I}} = j^{\mathcal{I}}$ holds. Thus, i and j are basically different *names* for the same domain individual in these models. Even if $i^{\mathcal{I}} = j^{\mathcal{I}}$ in all models, still $A \not\models \mathit{same\_as}(i, j)$ holds. Also note that the unique name assumption for A-box individuals is no longer enforced in Racer 1.8. A corresponding atom $\mathit{same\_domain\_individual}(i, j)$ will be added to a future nRQL version.

Now that we have defined truth of of positive ground query atoms, we can define truth of arbitrary ground query atoms:

Definition 6 (Truth of Ground Query Atoms). *Let A be a ground query atom. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base (T-box/A-box tuple). A ground atom A is either* TRUE *in $\mathcal{K}$ (we write $\mathcal{K} \models_{NF} A$) or* FALSE *in $\mathcal{K}$ (we write $\mathcal{K} \not\models_{NF} A$). The relationship $\models_{NF}$ resp. trueness of ground query atoms is inductively defined as follows:*

- *If A is positive (does not contain "$\backslash$"): $\mathcal{K} \models_{NF} A$ iff $\mathcal{K} \models A$*
- *Otherwise: $\mathcal{K} \models_{NF} \backslash(A)$ iff $\mathcal{K} \not\models_{NF} A$*

It is important to note that for each query body or atom q, q is TRUE iff $\backslash(q)$ is FALSE, and vice versa. Note that this does not hold for the usual entailment relationship. For example, consider the A-box $\{\mathit{woman}(\mathit{betty})\}$. Given $\mathcal{K} =_{def} (\emptyset, \mathcal{A})$, $\mathit{woman}(\mathit{betty})$ is TRUE, and $\mathit{mother}(\mathit{betty})$ is FALSE, since we cannot prove that *betty* is a mother. Thus, $\backslash(\mathit{mother}(\mathit{betty}))$ is TRUE. In contrast, $\neg\mathit{mother}(\mathit{betty})$ is obviously FALSE. Moreover, $(\mathit{mother} \sqcup \neg\mathit{mother})(\mathit{betty}) = \top(\mathit{betty})$ is *not* the same as $(\mathit{mother}(\mathit{betty})) \vee (\neg\mathit{mother}(\mathit{betty}))$.

In order to check whether $\mathcal{K} \models_{NF} A$, we can use the basic consistency checking and A-box retrieval methods offered by Racer. The symbol "$\models_{NF}$" shall remind the reader of the employed "Negation as Failure" semantics (i.e., suppose A is positive, then $\mathcal{K} \models_{NF} \backslash(A)$ iff $\mathcal{K} \not\models A$, which means $\backslash(A)$ is TRUE in $\mathcal{K}$, see below for examples).

The truth definition of ground atoms can be extended to complex *ground query bodies* in the obvious way (i.e., $\mathcal{K} \models_{NF} b_1 \wedge \cdots \wedge b_n$ iff $\forall b_i : \mathcal{K} \models_{NF} b_i$, and analogously for $\vee$ and $\backslash$).

Having defined truth of ground query atoms and bodies, we can specify the semantics of queries which are not ground, but first we need one more piece of notation. The rationale behind the next definition is best understood with an example: consider the query $\mathit{ans}(\mathit{betty}) \leftarrow \mathit{woman}(\mathit{betty})$. The answer to this query should either be $\emptyset$ (in case $\mathcal{K} \not\models \mathit{woman}(\mathit{betty})$), or $\{(\mathit{betty})\}$ (in case $\mathcal{K} \models \mathit{woman}(\mathit{betty})$). A reasonable statement is that $\mathit{ans}(\mathit{betty}) \leftarrow \backslash(\mathit{woman}(\mathit{betty}))$ should be the complement query of $\mathit{ans}(\mathit{betty}) \leftarrow \mathit{woman}(\mathit{betty})$. Thus,

$ans(betty) \leftarrow \backslash(woman(betty))$ should therefore return the set $\{\,(i) \mid i \in \mathsf{inds}(\mathcal{A})\,\}$ if $\mathcal{K} \not\models woman(betty))$ and $\{\,(i) \mid i \in \mathsf{inds}(\mathcal{A})\,\} \setminus \{betty\}$ if $\mathcal{K} \models woman(betty)$. Thus, within $\backslash(woman(betty))$, *betty* behaves in fact like a variable. To capture this behavior, we replace the individuals in the atoms with representative variables and use *same_as* statements as follows:

Definition 7 (α-Substitution). *Let A be an atom that contains at most one "$\backslash$" (note that $A = \backslash(\backslash(A))$). Denote the set of mentioned individuals in A as* $\mathsf{inds}(A)$. *Then, $\alpha(A)$ is defined as follows:*

- *If* $\mathsf{inds}(A) = \emptyset$*, then $\alpha(A) =_{def} A$.*
- *If A is* positive *and* $\mathsf{inds}(A) = \{i, j\}$ *(possibly $i = j$), then* $\alpha(A) =_{def} A_{[i \leftarrow x_i, j \leftarrow x_j]} \wedge same\_as(x_i, i) \wedge same\_as(x_j, j)$.
- *If $A = \backslash(A')$ is* negative *and* $\mathsf{inds}(A) = \{i, j\}$ *(possibly $i = j$), then* $\alpha(A) =_{def} \backslash(A'_{[i \leftarrow x_i, j \leftarrow x_j]}) \vee \backslash(same\_as(x_i, i)) \vee \backslash(same\_as(x_j, j))$.

Note that $A_{[i \leftarrow x_i, j \leftarrow x_j]}$ means "substitute i with x_i, and j with x_j". For example, $\alpha(R(i, j)) = R(x_i, x_j) \wedge same\_as(x_i, i) \wedge same\_as(x_j, j)$, but $\alpha(\backslash(R(i, j))) = \backslash(R(x_i, x_j)) \vee \backslash(same\_as(x_i, i)) \vee \backslash(same\_as(x_j, j))$. We extend the definition of α to query bodies in the obvious way. However, we need to bring the bodies into *negation normal form (NNF)* first, such that "$\backslash$" appears only in front of atoms. This is simply done by applying *DeMorgan's Law* to the query body (from the given semantics it follows that $\backslash(A \wedge B) \equiv \backslash(A) \vee \backslash(B)$, $\backslash(A \vee B) \equiv \backslash(A) \wedge \backslash(B)$, $\backslash(\backslash(A)) \equiv A$). The semantics of a nRQL query can now be paraphrased as follows:

Definition 8 (Semantics of a Query). *Let $ans(a_{i_1}, \ldots, a_{i_m}) \leftarrow body(a_1, \ldots, a_n)$ be a nRQL query q such that body is in NNF. Let $\beta(a_i) =_{def} x_{a_i}$ if $a_i \in \mathcal{I}$, and a_i otherwise; i.e., if a_i is an individual we replace it with its representative unique variable which we denote by x_{a_i}. Let $\mathcal{K}$ be the knowledge base to be queried, and $\mathcal{A}$ be its A-box. The* answer set *of the query q is then the following set of tuples:*

$$\{\,(j_{i_1}, \ldots, j_{i_m}) \mid \exists j_1, \ldots, j_n \in \mathsf{inds}(\mathcal{A}), \forall m, n, m \neq n : j_m \neq j_n,\\ \mathcal{K} \models_{NF} \alpha(body)_{[\beta(a_1) \leftarrow j_1, \ldots, \beta(a_n) \leftarrow j_n]}\,\}$$

Finally, we state that $\{()\} =_{def} \mathsf{TRUE}$ *and* $\{\} =_{def} \mathsf{FALSE}$.

Note that we assume the *unique name assumption (UNA)* for the variables here. However, the implemented query processing engine also offers non-UNA variables (prefixed with \$). For reasons of brevity we decided not to include them in the formal definitions in this paper. There are various other features in nRQL (which is still under development) which have been left out here. Please refer to [3] for the full list of nRQL features.

2.1 Example Session

In the following we discuss standard query patterns and discuss the expressivity of nRQL. As a running example we use the following simple knowledge base.

T-box:

$$\begin{aligned}
\mathit{has\_child} &\sqsubseteq \mathit{has\_descendant} \\
\mathit{inv\_has\_child} &\doteq \mathit{inv}(\mathit{has\_child}) \\
\mathit{has\_father} &\sqsubseteq \mathit{inv\_has\_child} \\
\mathit{has\_mother} &\sqsubseteq \mathit{inv\_has\_child} \\
\mathit{man} &\sqsubseteq \mathit{person} \\
\mathit{woman} &\sqsubseteq \mathit{person} \\
\mathit{brother} &\sqsubseteq \mathit{man} \\
\mathit{parent} &\doteq \mathit{person} \sqcap (\exists \mathit{has\_child}.\mathit{person}) \\
\mathit{mother} &\doteq \mathit{woman} \sqcap \mathit{parent} \\
\mathit{grandmother} &\doteq \mathit{mother} \sqcap \\
&\quad \exists \mathit{has\_child}.\exists \mathit{has\_child}.\mathit{person}
\end{aligned}$$

Role Declarations:

$$\begin{aligned}
&\mathit{transitive}(\mathit{has\_descendant}) \\
&\mathit{attribute}(\mathit{age}, \mathit{integer}) \\
&\mathit{feature}(\mathit{has\_father}) \\
&\mathit{feature}(\mathit{has\_mother})
\end{aligned}$$

A-box:

$$\begin{array}{lll}
\mathit{woman}(\mathit{alice}), & \mathit{woman}(\mathit{betty}), & \mathit{brother}(\mathit{charles}), \\
(\leq 1 \mathit{has\_sibling})(\mathit{charles}), & \mathit{has\_sister}(\mathit{eve}, \mathit{doris}), & \\
\mathit{has\_child}(\mathit{alice}, \mathit{betty}), & \mathit{has\_child}(\mathit{alice}, \mathit{charles}), & \mathit{has\_child}(\mathit{betty}, \mathit{doris}), \\
\mathit{has\_child}(\mathit{betty}, \mathit{eve}), & \mathit{has\_sibling}(\mathit{charles}, \mathit{betty}), & \mathit{has\_sister}(\mathit{doris}, \mathit{eve})
\end{array}$$

For example, $\mathit{ans}(x) \leftarrow \mathit{grandmother}(x)$ asks for all instances of type grandmother from the current A-box to be bound to the variable x. The answer is $\{(\mathit{alice})\}$. If we query with $\mathit{ans}(x) \leftarrow (\neg \mathit{grandmother})(x)$ we get $\{(\mathit{charles})\}$, since Charles is known to be a man. Due to the open world semantics, the other women in the A-box (Eve, Doris, Betty) might be grandmothers as well, and we just don't know. Consequently, $\mathit{ans}(x) \leftarrow \backslash(\mathit{grandmother}(x))$ yields $\{(\mathit{doris})(\mathit{eve})(\mathit{charles})(\mathit{betty})\}$, due to the negation as failure semantics. This is the complement of the first query.

If we just want to know if there are *any known grandmothers at all* in the current A-box, we can simply query with $\mathit{ans}() \leftarrow \mathit{grandmother}(x)$, and Racer replies TRUE.

As specified, it is possible to use A-box individuals within queries: $\mathit{ans}() \leftarrow \mathit{woman}(\mathit{betty})$ yields TRUE. Consequently, for $\mathit{ans}() \leftarrow \mathit{man}(\mathit{betty})$ the answer is FALSE. If A-box individuals (here *betty*) are used within a query head, they are listed in the bindings as well: $\mathit{ans}(\mathit{betty}) \leftarrow \mathit{woman}(\mathit{betty})$ yields $\{(\mathit{betty})\}$. However, for $\mathit{ans}(\mathit{betty}) \leftarrow \mathit{man}(\mathit{betty})$ the answer is again FALSE.

To give an example of a binary role query atom, consider $\mathit{ans}(\mathit{mother}, \mathit{child}) \leftarrow \mathit{has\_child}(\mathit{mother}, \mathit{child})$ which yields the answer set $\{(\mathit{betty}, \mathit{doris})(\mathit{betty}, \mathit{eve})$ $(\mathit{alice}, \mathit{betty})(\mathit{alice}, \mathit{charles})\}$. If we want to know who is not a child of whom, we use $\mathit{ans}(x, y) \leftarrow \backslash(\mathit{has\_child}(x, y))$ and get the complement of the previous query (minus tuples excluded due to the UNA for variables). If we are just interested in the children of Betty, we ask Racer with $\mathit{ans}(\mathit{child\_of\_betty}) \leftarrow \mathit{has\_child}(\mathit{betty}, \mathit{child\_of\_betty})$ and get $\{(\mathit{doris}), (\mathit{eve})\}$.

To give an example of a binary constraint query atom, consider $\mathit{ans}(x) \leftarrow\ > (\mathit{has\_father} \circ \mathit{has\_mother} \circ \mathit{age}(x), \mathit{has\_mother} \circ \mathit{has\_father} \circ \mathit{age}(x))$, which asks for persons whose father's mother is older than their mother's father.

Suppose we want to know for which individuals we have explicitly modeled children in the A-box. For this purpose, the query $\mathit{ans}(x) \leftarrow \mathit{has\_know\_}$

$successor(has\_child, x)$ can be used. Note that the main reason for having atoms of this type is related to negation: suppose we want to retrieve the A-box individuals which *do not* have a child. The query $ans(x) \leftarrow \backslash(has\_child(x, y))$ cannot be used, since first the complement of $has\_child(x, y)$ is computed, and then the projection to x is carried out. Thus, $ans(x) \leftarrow \backslash(has\_known\_successor(x, has\_child))$ must be used. We already noted that this query is not equivalent to $ans(x) \leftarrow \backslash(\exists has\_child.\top(x))$.

To illustrate the problem, suppose we want to query for *mothers* not having any explicitly modeled children in the A-box. Obviously, these mothers cannot be retrieved with $ans(x) \leftarrow \backslash(\exists has\_child.\top(x))$, since motherhood implies having a child. But this child need not be explicitly modeled in the A-box. Therefore, the query $ans(x) \leftarrow mother(x) \wedge \backslash(has\_known\_successor(x, has\_child))$ must be used. The syntax $ans(x) \leftarrow mother(x) \wedge has\_child(x, \texttt{NIL})$ is also understood by Racer.

To give an example of a more complex conjunctive query, suppose we are interested in all mothers of male persons: $ans(x, y) \leftarrow mother(x) \wedge man(y) \wedge has\_child(x, y)$. Racer replies: $\{(alice, charles)\}$. Due to the UNA for variables, the query $ans(x, y) \leftarrow man(x) \wedge man(y)$ returns $\emptyset$, since Charles is the only man. The same applies to $ans(x) \leftarrow man(x) \wedge man(charles)$, since x has to be bound to a man *different* from Charles.

We ask the reader to refer to the Racer manual [3] for examples of disjunctive queries and other nRQL peculiarities. In the following it will be sufficient to consider simple conjunctive queries whose variables range over A-box individuals.

3 Benchmarking Racer + nRQL

The so-called Lehigh University Benchmark (LUBM, [5]) was developed to facilitate the evaluation and comparison of OWL semantic web repositories. The LUBM consists of a OWL ontology modeling universities; i.e. concepts for persons, student, professors, publications, courses etc. as well as appropriate relationships for such a universe of discourse are modeled. A benchmark generator written in Java is capable to generate *extensional data* corresponding to this ontology; i.e. a set of departments, professors, students, courses, and so on is generated. A set of 13 benchmarking queries is defined, ranging from simple queries that can be answered using plain relational look-up techniques to more complicated queries which require dense OWL reasoning techniques in order to be answered completely. The LUBM queries are simple conjunctive queries referencing only concept and role names and can be mapped to nRQL. Please refer to [5] for more information about the queries. Figure 1 shows LUBM queries 9 and 12 - note that `www.University0.edu` is an individual, and `subOrganizationOf` is a transitive role.

In [5], the system DLDB, which is a relational database system (Microsoft Access) augmented with a DL-based query rewriting engine, is benchmarked according to this LUBM and it is observed that the queries 11, 12 and 13 cannot be completely answered with the current DLDB implementation, i.e. in its current state of development, DLDB is an incomplete OWL semantic web repository.

```
Q9: (retrieve
      (?x ?y ?z)
      (and (?x Student)
           (?y Faculty)
           (?z Course)
           (?x ?y advisor)
           (?x ?z takesCourse)
           (?y ?z teacherOf)))
```

```
Q12: (retrieve
       (?x ?y www.University0.edu)
       (and (?x chair)
            (?y Department)
            (?x ?y memberOf)
            (?y www.University0.edu
                subOrganizationOf)))
```

Fig. 1. LUBM Queries 9 und 12

For our benchmark, we used a LUBM repository comprising 1 university with 14 departments. We then ran the 13 queries on repositories of increasing size (on a P4 2.8 GHz 1 GB RAM machine running Linux) starting from a university with 1 department, and adding departments one by one to the repository. The answer time of each query is measured 10 times on repositories of increasing size, and the average answer time is recorded.

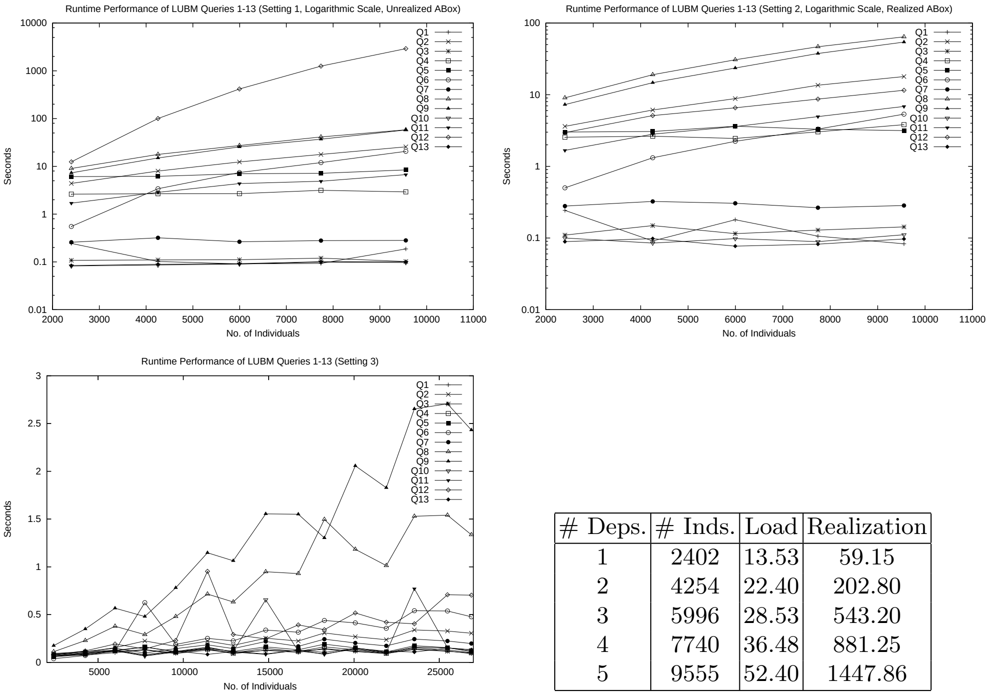
We ran the benchmark with three different settings: in setting 1, the A-box is *not realized* before queries are answered which means that the set of concept names an individual is an instance of is not computed before querying starts. In setting 2, the A-box is *first realized.* Setting 3 refers to a new incremental two-phase tuple-at-a-time query processing mode in Racer 1.8 (see below for an explanation). The A-box sizes, loading and realization time are also given in Figure 2.

With the current Racer version (1.8) it is not yet possible to load a whole university from the LUBM due to the initial A-box consistency test which has to be performed before query answering starts; we therefore had to stop at 5 departments.

From Figure 2 (note the logarithmic scale), we can observe the following: in setting 1, queries 1, 3, 7, 10 and 13 can be answered within fractions of a second, even if 5 departments are loaded. These queries as well as queries 4 and 5 show constant runtime behavior. This is not surprising, since they refer to individuals from the A-box, and are therefore very specific. For the queries 2, 8, 9, 11 and 12 an *exponential increase* in answering time is suggested. However, only queries 8, 9 and 12 need more than 55 seconds when 5 departments are loaded. These are the hard ones in setting 1. Query 12 needs even 2912 seconds.

In setting 2, with a realized A-box, the situation changes: query 12 can now be answered in 12 seconds! Queries 1, 3, 4, 5, 7, 10 and 13 still show constant behavior and run only slightly faster. It is also suggested that the exponential increase for queries 2, 8 and 9 is somehow slightly alleviated (more sampling points are needed in order to be more concise about this). However, only query 12 *really* benefits from setting 2. The overall execution times for the other queries are only slightly faster.

The analysis suggests that using a realized A-box can alleviate an exponential increase in query answering time. However, prior realization only pays off when queries are posed that need deep reasoning capabilities for being answered



# Deps.	# Inds.	Load	Realization
1	2402	13.53	59.15
2	4254	22.40	202.80
3	5996	28.53	543.20
4	7740	36.48	881.25
5	9555	52.40	1447.86

Fig. 2. Runtime Behavior of LUBM Queries - Setting 1, 2, 3

completely. Note that DLDB could not answer query 12. A-box realization times increase quite drastically, as Figure 2 shows. Queries 8 and 9 are hard in both settings. Query 12 becomes much easier in setting 2. Query 8 and 9 are also among the hardest queries for DLDB [5].

Let us now discuss setting 3. Since version 1.8, Racer supports incremental query answering. Thus, rather than retrieving the whole solution set in a single step (set-at-a-time approach), client applications can load elements from the solution set in a way that proceeds tuple by tuple (tuple-at-a-time iterator approach). Additionally, Racer 1.8 can be configured in such a way that nRQL queries are answered with optimized algorithms known from relational databases. Thus, similar to DLDB, Racer treats an A-box as a database and rewrites queries in order to support T-box information for information retrieval. In addition to DLDB, Racer also supports inverse roles in this process. In Figure 2 setting 3 the times for retrieving all tuples that are also found by DLDB (plus some more due to inverse roles) are indicated. Racer 1.8 supports a query function that indicates when the last of these "cheap" answers is returned. After this point, Racer switches to full A-box mode and retrieves the remaining tuples. However, much more computational resources are required (see the discussion above). It is up to the application to decide if it is worth the effort. Figure 2 setting 3 indicates that Racer 1.8 provides comparable performance for similar services as offered by hybrid systems such as DLDB, but offers complete reasoning if required.

4 Related Work, Discussion & Conclusion

For querying OWL semantic web repositories, the query language OWL-QL has been proposed [6]. An OWL-QL query is basically a full OWL KB together with a specification which of the URIs referred to in the query pattern are to be interpreted as variables. Variables come in three forms: *must-bind, may-bind, and do-not bind variables.* OWL-QL uses the standard notion of logical entailment: query *answers* can be seen as *logically entailed sentences* of the queried KB. Unlike in nRQL, variables cannot only be bound to constants resp. explicitly modeled A-box individuals, but also to complex *OWL terms* which are meant to denote the logically implied domain individual(s) from $\Delta^{\mathcal{I}}$. Thus, if variables in the query patterns are substituted with answer bindings, the resulting sentences are logically entailed by the queried KB. For must-bind variables, bindings *have* to be provided. May-bind variables may provide bindings or not, and *do-not-bind variables* are purely existentially quantified ("existential blanks in the query"). Moreover, OWL-QL queries are full OWL KBs, and this implies that not only extensional queries like in nRQL must be answered, but also "structural queries" are possible, such as "retrieve the subsuming concept names of the concept name father". Similar functions are also offered by Racer's API, but are not available in nRQL However, nRQL is not really a subset of OWL-QL. In OWL-QL, neither negation as failure nor *disjunctive A-boxes* can be expressed. Moreover, binary constraint query atoms of nRQL as well as negated has-known-successor query atoms "are missing" in OWL-QL. The latter ones have in fact been requested by the first users of the nRQL implementation. This might indicate that a limited kind of *autoepistemic or closed-world query facilities* should be present in a DL query language.

In this paper we have presented the Racer + nRQL semantic web repository, as well as some preliminary benchmarking results.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Haarslev, V., Möller, R.: Racer system description. In: International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy. (2001)
3. Haarslev, V., Möller, R.: The Racer user's guide and reference manual (2003)
4. Haarslev, V., Möller, R.: Optimization techniques for retrieving resources described in owl/rdf documents: First results. In: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004), Whistler (Canada) (2004)
5. Guo, Y., Heflin, J., Pan, Z.: Benchmarking DAML+OIL repositories. In: Proc. of the Second Int. Semantic Web Conf. (ISWC 2003). Number 2870 in Lecture Notes in Computer Science, Springer-Verlag (2003) 613–627
6. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL - a language for deductive query answering on the semantic web. Technical Report KSL-03-14, Knowledge Systems Lab, Stanford University, CA, USA (2003)