Tasks for Ontology Access, Processing, and Usage

Deliverable TONES-D10

Diego Calvanese[1], Enrico Franconi[1], Birte Glimm[3], Bernardo Cuenca Grau[3], Ian Horrocks[3], Alissa Kaplunova[5], Domenico Lembo[2], Maurizio Lenzerini[2], Carsten Lutz[4], Ralf Möller[5], Riccardo Rosati[2], Ulrike Sattler[3], Sergio Tessaris[1], Anni-Yasmin Turhan[4]

[1] Free University of Bozen-Bolzano, [2] Università di Roma "La Sapienza",
[3] The University of Manchester, [4] Technische Universität Dresden,
[5] Technische Universität Hamburg-Harburg

Project:	FP6-7603 – Thinking ONtolgiES (TONES)
Workpackage:	WP4– Ontology access, processing, and usage
Lead Participant:	Hamburg University of Technology
Reviewer:	Giuseppe de Giacomo
Document Type:	Deliverable
Classification:	Public
Distribution:	TONES Consortium
Status:	Final
Document file:	D10_TasksAcessProcessingUsage.pdf
Version:	1.1
Date:	August 31, 2006
Number of pages:	80

Abstract

Research about ontology access, processing, and usage paves the way for realizing important tasks in future applications requiring well-understood formal representation formalisms as well as efficient and industrial-strength implementations. In this report, we summarize the state of the art for most important application tasks of this kind that use ontologies as their backbone. In addition to a formalization of the tasks for some of the most important application scenarios, we also report on recent theoretical and practical advances for their realization that have been achieved as part of our work in the TONES project.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	August 10, 2006	First draft
v.1.1	August 31, 2006	Final version

Contents

Preface

For building applications, ontologies can be used for various purposes. On the one hand, they can be used to verify the conceptual data model (see, e.g., [13]), which then can be used to build a standard application using well-known software-engineering techniques. On the other hand, ontologies can also be used to actually solve (sub-)problems occurring in application tasks if these (sub-)problems can be formalized using decision problems defined for a certain ontology framework. In particular, in the context of the Semantic Web one of the main tasks, information access and retrieval, can be formalized using ontologies. In this report, we analyze tasks involving ontology access, processing, and usage in order to demonstrate the state of the art. We also investigate remaining problems, identify challenges, and present first results that have been achieved in the TONES project.

In the following, we focus on ontology languages that are based on description logics, one of the major fragments of first-order logic for which decidable decision problems related to query answering and other ontology-based tasks can be defined.

The report is subdivided into two main parts. After this introduction, we investigate ontology-based information access as a first task in which ontologies are used to solve application problems. Assuming that information is explicitly and declaratively specified we discuss practical ontology-based query answering systems, demonstrate ontology and query languages tailored towards specific mass-data requirements, analyze their complexity, and present query answering algorithms w.r.t. increasing expressiveness of the ontology definition language. In addition, we analyze how an ontology system could support a user in formulating queries for certain information needs. The proposed techniques work if information to be retrieved is explicitly represented (in terms of data descriptions or in terms of declarative meta information already attached to information objects). However, even in the context of the Semantic Web it might not always be the case that meta information is available for information represented as images, videos, audio files, or even natural language documents. We present a formalization of an information extraction approach that uses ontologies to extract information from media data and makes it automatically available as meta data for media such that the aforementioned query languages can be used to retrieve these documents containing relevant information. We also investigate how ontologies can be used to manipulate indefinite information appropriately. Starting from standard syntactic approaches we also cover "semantic updates" of information sources. Afterwards, in the last part it is explained how ontology systems can be used to formalize and solve other specific tasks in applications. In order to demonstrate the main insights of how ontologies can be used to solve problems, we focus on semantic service discovery and selection as well as on how to use ontologies to construct or configure technical devices.

Part I
Ontology-based Information Access and Manipulation

1 Query Answering

Ontology-based information access is the predominant example that demonstrates why recently ontologies have become very popular in many scientific and practical fields. With the expressive power of ontologies, queries can be posed w.r.t. the vocabulary of the user rather than have to be specified w.r.t. the low-level vocabulary of the data model of a particular information source. Depending on the expressiveness, we distinguish between single- and multi-model ontologies. The latter kind can express indefinite information and, speaking in extensional terms, we talk of data descriptions (which can have multiple models) rather than merely data (which represent a single model). For information access we distinguish two different scenarios:

1. Data descriptions are already available, i.e. query answering is to be defined w.r.t. explicitly given data or data descriptions. Depending on the context, two different aspects are important for query answering:

    - Focus is on inferred information (the standard view, in which there is no distinction between explicitly given information and derived information);
    - Focus is on explicitly stated information only (aka told information access, a kind of ontology processing which is relevant, e.g., for developing ontology design tools and for using ontology processing capabilities of existing DL systems for implementing algorithms for solving higher-level decision problems).

2. Information is implicit in media data, and logical descriptions of the content must be derived in beforehand. This concerns, for instance, query w.r.t. media data for which an information extraction process is required in order to make ontologies apply to query answering.

Informally speaking, query answering with respect to an ontology means to find tuples of individuals that satisfy certain conditions. For specifying conditions, various kinds of query languages with different expressivity have been investigated in the literature (e.g., first-order logic, datalog [24], etc.). In this report we focus on query languages in the context of ontologies based on description logics. Syntax and semantics of specific languages will be introduced in subsequent sections.

We start with practical DL systems and present the main ideas of query languages based on so-called grounded conjunctive queries that are currently supported by system implementations and analyze their capabilities and shortcomings. Hereafter the report presents theoretical work on query languages supporting (full) conjunctive queries for very expressive ontology languages, and then analyzes conjunctive queries with first-order reducibility for less expressive ontology languages.

1.1 Practical Query Answering in DL Systems

In many description logic application contexts (e.g., Semantic Web, natural language and image understanding, software verification, etc.), problems are reduced to ABox inference problems and, hence, an ABox query language is required. Concept-based retrieval of individuals as provided by older description logic systems is often not enough in many application scenarios.

For practical systems, standards are of utmost importance. Users will only invest in a certain technology if they do not have to rely on proprietary languages. For research purposes, however, standards can also be an obstacle because new features, which are required for applications, cannot be supported easily (except with non-standard extensions to some existing standard). While there exist standards for query languages such as OWL-QL [28], in many application projects the expressivity offered by this language is not enough because, for instance, negation as failure inferences are used in many applications but are not supported in OWL-QL. In this section we study a very expressive query language that goes beyond OWL-QL in some respects but has less expressivity in others in order to provide for efficient implementation. The scientific problem is to specify the semantics of such a language such that (i) it is practically useful (i.e., expressivity meets practical requirements), (ii) language constructs can be combined in an orthogonal way, and (iii) a coherent set of basic query atoms is provided in such a way that a semantics can be specified in a declarative way (i.e., special-purpose language constructs can be introduced as syntactic sugar but need not to be treated with specific techniques). In order to support practical experiments, the language is implemented as part of the RacerPro description logic system [39]. The language is called the *n*ew *R*acer *Q*uery *L*anguage, nRQL (pronounce "nercle"), which now is an integral part of the RacerPro 1.9 engine. nRQL was designed with a focus on conceptual simplicity as well as language orthogonality on the one hand, but also meets the requirement to offer full access to all ABox features available in Racer, for example, it provides access to the concrete domain part of an ABox [7]. Since its first release [42], nRQL has developed considerably. The insights behind the language design are summarized in this section. We contribute to further standardization efforts, which might select a specific, XML-based surface syntax also for those features not covered by OWL-QL. For instance, in [31] we investigate how subsets of nRQL can be used to implement an OWL-QL query answering engine for the subset of so-called grounded conjunctive queries (see below). We use the name OWL-QL$^-$ in order to denote OWL-QL with grounded conjunctive queries.

Since *extensional information* from OWL documents (OWL instances and their interrelationships) are represented with ABoxes (with associated TBoxes), it is apparent that nRQL can also be understood as a *Semantic Web query language.* In order to support special OWL features such as *annotation* and *datatype properties*, special OWL querying facilities have been incorporated into nRQL.

This paper is structured as follows. First, we informally describe the main ideas behind the language constructs of the nRQL language. Then we formally specify the syntax and semantics of nRQL. After that, the main ideas of the nRQL query answering engine (i.e., the implementation of query semantics) are presented.

1.1.1 Motivation of the nRQL Language

In the following we describe nRQL's main language features as well as the core design principles we have followed. We use a syntax tailored towards interactively posing queries rather than consider a verbose XML-based syntax designed for machine processing. As usual, a nRQL query consists of a *query head* and a *query body*. For example, the query

```
(retrieve (?x ?y) (and (?x woman) (?x ?y has-child)))
```

has the head `(?x ?y)` and the body `(and (?x woman) (?x ?y has-child))`. It returns all mother-child individuals pairs from the ABox. In a nutshell, the nRQL *language* (to be distinguished from the nRQL *engine*, see below) can be characterized as follows:

Variables and *individuals* can be used in queries. The variables range over the individuals of an ABox, and are bound to those ABox individuals which *satisfy* the query. The notion of satisfiability of a query used in nRQL is defined in terms of logical entailment. This means, a variable is bound to an ABox individual iff it can be proved that this binding holds in *all* models of the knowledge base. Because variables are bound only to individuals mentioned in a knowledge base (and not to individuals that might exist due to existential restrictions), we call nRQL a *grounded* query language (the semantics is also called *active domain semantics* [1]).

Returning to our example query body `(and (?x woman) (?x ?y has-child))`, `?x` is only bound to those individuals which are instances of the concept `mother` having a *known* child `?y` in *all* models of the KB.

nRQL distinguishes variables which must be bound to *differently named individuals* (prefix `?`, e.g., `?x, ?y` cannot be bound to the same ABox individual) from variables for which this does not hold (prefix `$?`, e.g., `$?x, $?y`). Individuals from an ABox are identified by using them directly in the query, e.g. `betty`.

Different types of query atoms are available: these include concept query atoms, role query atoms, constraint query atoms, and same-as query atoms. To give some examples, the atom `(?x (and woman (some has-child female)))` is a concept query atom, `(?x ?y has-child)` is a role query atom, `(?x ?x (constraint (has-father age) (has-mother age) =))` is a constraint query atom (asking for the persons `?x` whose parents have equal age), and `(same-as ?x betty)` is a same-as query atom, enforcing the binding of `?x` to `betty`.

As the given example concept query atom demonstrates, it is possible to use complex concept expressions within concept query atoms. Regarding role query atoms, the set of role expressions is more limited. However, it is possible to use inverted roles (e.g., role expressions such as `(inv R)`) as well as *negated roles* within role query atoms. For example, the atom `(?x ?y (not has-father))` will return those bindings for `?x, ?y` for which Racer *can prove* that the individual bound to `?x` cannot have the individual bound to `?y` as a father. If the role `has-father` was defined as having the concept `male` as a range, then at least all pairs of individuals in which `?y` is bound to a `female` person are returned, given `male` and `female` can be proved to be disjoint.

Complex queries are built from query atoms using the boolean constructors `and, union, neg`. We have already seen an example: `(and (?x woman) (?x ?y has-child))` is a simple *conjunctive query body.* These constructors can be combined in an arbitrary way to create complex queries. This is why we call nRQL an orthogonal language.

The `neg` operator implements a *negation as failure semantics (NAF)*: `(neg (?x woman))` returns all ABox individuals for which Racer *cannot prove* that they are instances of `woman`. Thus, `(neg (?x woman))` returns the complement set of `(?x woman)` w.r.t. the set of all ABox individuals. If used in front of a role query atom, e.g. `(neg (?x ?y has-child))`, this returns the two-dimensional set difference of `(?x ?y has-child)` w.r.t. the Cartesian product of all ABox individuals, i.e. all pairs of individuals which are not in the extension of `has-child` in all models.

The semantics of nRQL ensures that DeMorgan's Laws hold:

`(neg (and` $a_1 \ldots a_n$`))` is equivalent to `(union (neg` a_1`)` ... `(neg` a_n`))`, and that the set difference using `neg` is well-defined for each basic query atom.

Please note that `(?x (not woman))` has a different semantics from `(neg (?x woman))`, since the former returns the individuals for which Racer *can prove* that they are *not instances* of woman, whereas the latter returns all instances for which Racer *cannot prove* that they are *instances* of woman. The same line of argumentation applies to role and constraint query atoms, although NAF negation of constraint query atoms is more involved in the presence of *role chains*.

Support for retrieving told concrete domain values *from the concrete domain part of a Racer ABox:* Suppose that `age` is a so-called *concrete domain attribute* of type integer. Thus, the `age` attribute fillers of a certain individual must be concrete domain values of type `integer`. We can use the following query to retrieve all adults as well as their ages: `(retrieve (?x (told-value (age ?x)) (?x (min age 18))))`, and a possible answer might be `(((?x michael) ((told-value (age michael)) 34)))`.

Since nRQL variables are bound to ABox individuals but not to concrete domain datatype values (for reasons of safeness, see below), nRQL includes a special set of so-called *head projection operators* in order to support retrieval of concrete domain datatype values. These operators are denoted in a functional style in the head of a query - `(told-value (age ?x))` is such an operator. Moreover, a concrete domain value such as `34` can only be retrieved if it has been *told* to Racer - that is, the value must be explicitly (syntactically) specified in the ABox. The rationale for this is that description logic systems do not compute the solutions of a concrete domain constraint system, but only decide its satisfiability. There would be no way to return these solutions as bindings to variables in the general case. Therefore, nRQL does not offer variables which range over concrete domain values. Allowing so would either destroy the orthogonality of nRQL, or make the language unsafe - after all, given a query such as `(retrieve (?x ?y) (and (?x ?y age) (?y (and integer (min 18)))))` would result in an infinite set of binding tuples. Note that concrete domain attributes are not roles; thus, `(?x ?y age)` is syntactically invalid. For these reasons, querying for concrete domain values is only supported by means of concept expressions and head projection operators.

Moreover, constraint query atoms allow one to "compare" concrete domain attribute fillers of different individuals. Consider the query

```
(retrieve (?x (told-value (age ?x)))
      (and (?x (and woman (an age))) (?x ?y has-child)
           (?y ?y (constraint (has-father age) (has-mother age)
                              (< (+ age-1 8) age-2)))))
```

which returns the list of women and their ages. The women are required to have children whose fathers are at least 8 years older than their mothers. Note that `(has-father age)` denotes a "path expression": starting from the individual bound to `?y` we retrieve the value of the concrete domain attribute `age` of the individual which is the filler of the `has-father` role (feature) of this individual. In a similar way, the age of the mother of `?y` is retrieved. These concrete domain values are then used as actual arguments to evaluate the compound concrete domain predicate `(< (+ age-1 8) age-2)`. Here, `age-1` refers to `(has-father age)`, and `age-2` refers to `(has-mother age)`. Note that the suffixes `-1, -2` have been added to the `age` attribute in order to differentiate the two values. Obviously, this mechanism is not needed in case the two chains are ended by different attributes.

Special support for querying OWL documents, e.g., retrieving told datatype value fillers of OWL datatype and OWL annotation properties. Retrieval of these datatype values is supported in a similar style as in the concrete domain case, by means of concept query atoms and head projection operators.

Since concept expressions such as `(min age 18)` are useful in concept query atoms for specifying constraints on the datatype values to be retrieved, we have extended the Racer concept expression syntax to allow for similar concept query atoms containing datatype properties instead of attributes. This means, the concrete domain constraint expression language of Racer has been extended to also cover OWL datatype properties in addition to attributes. Thus, if `age` is not a concrete domain attribute, but now an OWL datatype property, then the previous query would work as well. Using concept query atoms, we can also query for the (non)existence of fillers of annotation properties of individuals. This means, if `AP` is an annotation property, then only concept query atoms such as `(?x (an AP))` and `(?x (no AP))` are defined. The filler annotation values of these individuals `?x` can indeed be retrieved by means of the `annotations` head projection operators as well (see syntax specification).

nRQL also offers a body projection operator. Sometimes this operator is required in order to reduce the "dimensionality" of a tuple set, for example, before computing a set difference with an *n*-dimensional set. This is exactly what happens if `neg` is used.

Let us motivate the necessity for such an operator: consider `(retrieve (?x) (and (?x mother) (?x ?y has-child)))`. This query returns all mothers having a *known child* in the ABox. Now, how can we query for mothers which do *not* have a *known* child?

Our first attempt will be the query `(retrieve (?x) (and (?x mother) (neg (?x ?y has-child))))`. A bit of thought and recalling that `(neg (?x ?y has-child))` returns the complement set of `(?x ?y has-child)` w.r.t. the Cartesian product of all ABox individuals will reveal that this query doesn't solve the task. In a second attempt will would probably try `(retrieve (?x) (neg (and (?x mother) (?x ?y has-child))))`. However, due to DeMorgan's Law and nRQL's semantics, this query is equivalent to `(retrieve (?x) (union (and (neg (?x mother)) (?y top)) (neg (?x ?y has-child))))` – first the union of two two-dimensional tuple sets is constructed, and then only the projection to the first element of these pairs (`?x`) is returned. Obviously, this set contains also the instances which are *not* known to be mothers, what is wrong as well. Thus, the need for the projection operator becomes apparent: `(retrieve (?x) (and (?x mother) (neg (project-to (?x) (?x ?y has-child)))))` solves the task. This body projection operator was not present in earlier versions of nRQL, special syntax was introduced to address these problems, namely the unary special atoms `(?x (has-known-successor has-child))`, `(?x NIL has-child)` and `(NIL ?x child-of)`. These atoms (which still work) can now be seen as "syntactic sugar" for the bodies `(project-to (?x) (?x ?y has-child))`, `(neg (project-to (?x) (?x ?y has-child)))` and `(neg (project-to (?x) (?y ?x has-child)))`. The `project-to` operator can be used at any position in a query body.

nRQL also offers defined queries. These can be understood as a simple macro mechanism. For example, the form `(defquery mother (?x) (and (?x woman) (?x ?y has-child)))` can be used to define a query `mother` which can be subsequently used in calls such as `(retrieve (?x) (?x mother))` or in subsequent definitions, e.g. `(defquery married-mother (?x) (and (?x mother) (?x ?y has-spouse)))`. The definitions must be acyclic.

1.1.2 Syntax and Semantics of nRQL

Definition 1 (Syntax of nRQL)**.** A nRQL query has a *head* and a *body*. A *head* can contain the following (note that {a|b} represents a or b):

head	:=	(*head_entry**)
object	:=	*variable* | *individual*
variable	:=	a symbol beginning with "?"
individual	:=	a symbol
head_entry	:=	*object* | *head_projection_operator*
head_projection_operator	:=	(*cd_attribute object*) |
		(`told-value` (*cd_attribute object*)) |
		(`told-value` (*datatype_property object*)) |
		(`annotations` (*annotation_property object*))

The *body* is defined as follows:

body	:=	*atom* | ({`and` | `union`} *body**) | (`neg` *body*) |
		(`project-to` (*object**) *body*)
atom	:=	(*object concept_expr*) | (*object object role_expr*) |
		(*object object* (`constraint` *chain chain constraint_expr*)) |
		(`same-as` *object object*)
chain	:=	(*role_expr** *cd_attribute*)

The "bridge" to the Racer syntax is given by the following rules:

concept_expr	:=	a Racer concept, with some extensions for OWL
role_expr	:=	a Racer role or the *special role* `equal` |
		(`inv` *role_expr*) | (`not` *role_expr*)
constraint_expr	:=	a (possibly compound) Racer concrete domain
		predicate, e.g. `(< (+ age-1 20) age-2)`
cd_attribute	:=	a Racer concrete domain attribute
datatype_property	:=	a Racer role used as OWL datatype property

△

Note that nRQL allows the use of negated roles in role atoms, as well as the special role `equal`. The semantics of `equal` is fixed: $\texttt{equal}^{\mathcal{I}} =_{def} \mathcal{ID}_{2,\Delta^{\mathcal{I}}}$, (see below for the definition of $\mathcal{ID}$); thus, `equal` is interpreted as the two-dimensional identity relationship.

Definition 2 (Semantics of nRQL)**.** Let $\mathcal{A}$ be a Racer ABox, and $\mathcal{T}_{\mathcal{A}}$ denote its associated TBox. Denote the set of individuals used in $\mathcal{A}$ with $\mathsf{Inds}_{\mathcal{A}}$.

Let q be a nRQL query *body*. The function $\mathsf{vars}(q)$ is defined inductively:
$\mathsf{vars}(\texttt{(}x\ \mathit{concept\_expr}\texttt{)}) =_{def} \{x\}$,
$\mathsf{vars}(\texttt{(}x_1\ x_2\ \mathit{role\_expr}\texttt{)}) =_{def} \{x_1, x_2\}$,
$\mathsf{vars}(\texttt{(}x_1\ x_2\ \texttt{(constraint ...))}) =_{def} \{x_1, x_2\}$,
$\mathsf{vars}(\texttt{(\{ and | union | neg \}}\ q_1 \ldots q_m\texttt{)}) =_{def} \bigcup_{1 \le i \le m} \mathsf{vars}(q_i)$, **BUT**
$\mathsf{vars}(\texttt{(project-to (}x_1 \ldots x_m\texttt{) ...)}) =_{def} \{x_1 \ldots x_m\}$.

Thus, `vars` "stops at projections". Assume that $\langle x_{1,q}, \ldots, x_{n,q}\rangle$ is a lexicographic enumeration of $\mathsf{vars}(q)$. Denote the ith element in this vector with $x_{i,q}$, indicating its position in the vector.

Let $\mathcal{T}$ be a set of n-ary tuples $\langle t_1, \ldots, t_n\rangle$ and $\langle i_1, \ldots, i_m\rangle$ be an *index vector* with $1 \le i_j \le n$ for all $1 \le j \le m$. Then we denote the set $\mathcal{T}'$ of m-ary tuples with $\mathcal{T}' =_{def} \{ \langle t_{i_1}, \ldots, t_{i_m}\rangle \mid \langle t_1, \ldots, t_n\rangle \in \mathcal{T} \} = \pi_{\langle i_1,\ldots,i_m\rangle}(\mathcal{T})$, called the *projection* of $\mathcal{T}$ to the components mentioned in the index vector $\langle i_1, \ldots, i_m\rangle$. For example, $\pi_{\langle 1,3\rangle}\{\langle 1,2,3\rangle, \langle 2,3,4\rangle\} = \{\langle 1,3\rangle, \langle 2,4\rangle\}$.
Let $\vec{b} = \langle b_1, \ldots, b_n\rangle$ be a *bit vector* of length n, $b_i \in \{0,1\}$. Let $m \le n$. If $\vec{b}$ is a bit vector which contains exactly m ones, and $\mathcal{B}$ is a set, $\mathcal{T}$ is a set of m-ary tuples, then the n-dimensional *cylindrical extension* $\mathcal{T}'$ of $\mathcal{T}$ w.r.t. $\mathcal{B}$ and $\vec{b}$ is defined as

$$\begin{aligned}\mathcal{T}' =_{def} \{ \langle i_1, \ldots, i_n\rangle \mid\ & \langle j_1, \ldots, j_m\rangle \in \mathcal{T}, 1 \le l \le m, 1 \le k \le n \\ & \text{with } i_k = j_l \text{ if } b_k = 1, \\ & \text{and } b_k \text{ is the } l\text{th one (1) in } \vec{b}, \\ & \text{otherwise, } i_k \in \mathcal{B} \ \ \}\end{aligned}$$

and denoted by $\chi_{\mathcal{B},\langle b_1,\ldots,b_n\rangle}(\mathcal{T})$. For example,
$\chi_{\{a,b\},\langle 0,1,0,1\rangle}(\{\langle x,y\rangle\}) = \{\langle a,x,a,y\rangle, \langle a,x,b,y\rangle, \langle b,x,a,y\rangle, \langle b,x,b,y\rangle\}$.

We denote an n-dimensional bit vector having ones at positions specified by the index set $\mathcal{I} \subseteq 1 \ldots n$ as $\vec{\mathbf{1}}_{n,\mathcal{I}}$. For example, $\vec{\mathbf{1}}_{4,\{1,3\}} = \langle 1,0,1,0\rangle$. Moreover, with $\mathcal{ID}_{n,\mathcal{B}}$ we denote the n-dimensional identity relation over the set $\mathcal{B}$: $\mathcal{ID}_{n,\mathcal{B}} =_{def} \{ \underbrace{\langle x, \ldots, x\rangle}_{n} \mid x \in \mathcal{B} \ \}$.

The semantics of a nRQL query is given by the set of tuples it returns if posed to an ABox $\mathcal{A}$. This set of answer tuples is called the extension of q and denoted by $q^{\mathcal{E}}$. We claim that the given semantics in terms of "tuple spaces" is substantially easier to grasp for users than the FOPL-based semantics (i.e., the one we gave for the older versions of nRQL in [42, 43]).

In order to simplify the specification of the semantics, the query body q first undergoes some syntactical transformations: In a first step, q is rewritten by consistently replacing all its individuals with new representative fresh variables throughout the body. If the individual i has been replaced with the variable x_i, then we also add the conjunct `(same-as` x_i i`)` to q, yielding a body of the form `(and` q `(same-as` x_i i`) (same-as ...) ...)`. In a second step, the *role chains* possibly present in constraint query atoms are decomposed. This means they are replaced by conjunctions of role query atoms such that only concrete domain attributes remain in chains of constraint query atoms. Fresh anonymous variables are used for this purpose. E.g., the atom `(?x ?y (constraint (has-father age) (has-mother age) =))` is replaced with the body

```
    (and (?x ?ano1-x-father has-father)
         (?y ?ano2-y-mother)
         (?ano1-x-father ?ano2-y-mother
             (constraint age age =))).
```

Let q' be the transformed query.

We can now define the semantics of the different query atoms, being part of q'. Keep in mind that $\langle x_{1,q'}, \ldots, x_{n,q'} \rangle$ is the variable vector of q' and that n is the total number of variables returned by $\mathsf{vars}(q')$. For the sake of brevity we only consider variables which do not prevent the binding of an ABox individual to multiple variables (but the semantics can be easily extended). Thus, the semantics for the different nRQL query atoms is given as:

$$
\begin{aligned}
(q'_{x_i}\ \mathit{concept\_expr})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{concept\_instances}(\mathcal{A}, \mathit{concept\_expr}))\\
(q'_{x_i}\ q'_{x_j}\ \mathit{rolen\_expr})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{role\_pairs}(\mathcal{A}, \mathit{role\_expr})),\ \text{if } i \neq j\\
(q'_{x_i}\ q'_{x_i}\ \mathit{role\_expr})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{role\_pairs}(\mathcal{A}, \mathit{role\_expr})\ \cap\ \mathcal{ID}_{2,\mathsf{Inds}_\mathcal{A}})\\
(\texttt{same-as}\ q'_{x_i}\ \mathit{ind})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\{\mathit{ind}\})\\
(\texttt{same-as}\ \mathit{ind}\ q'_{x_i})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\{\mathit{ind}\})\\
(\texttt{same-as}\ q'_{x_i}\ q'_{x_j})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathcal{ID}_{2,\mathsf{Inds}_\mathcal{A}}),\ \text{if } i \neq j\\
(\texttt{same-as}\ q'_{x_i}\ q'_{x_i})^{\mathcal{E}} &=_{def}\ \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\mathcal{ID}_{2,\mathsf{Inds}_\mathcal{A}})
\end{aligned}
$$

$$
\begin{aligned}
&(q'_{x_i}\ q'_{x_j}\ (\texttt{constraint}\ \mathit{attrib}_1\ \mathit{attrib}_2\ P))^{\mathcal{E}} =_{def}\\
&\qquad \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{predicate\_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P)),\ \text{if } i \neq j\\
&(q'_{x_i}\ q'_{x_i}\ (\texttt{constraint}\ \mathit{attrib}_1\ \mathit{attrib}_2\ P))^{\mathcal{E}} =_{def}\\
&\qquad \chi_{\mathsf{Inds}_\mathcal{A},\vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{predicate\_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P)\ \cap\ \mathcal{ID}_{2,\mathsf{Inds}_\mathcal{A}})
\end{aligned}
$$

This definition uses some auxiliary functions, providing the bridge to the classical ABox retrieval functions offered by Racer. These bridge functions have the standard DL semantics in terms of logical entailment. However, as already mentioned, a nRQL *role expression (role_expr)* can also be a *negated* (or inverse) role. In case of $\mathit{role\_expr} = \mathtt{equal}$ we have $(\mathcal{A}, \mathcal{T}_\mathcal{A}) \models \mathtt{equal}(i, j)$ iff $i^\mathcal{I} = j^\mathcal{I}$ in all models $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of $(\mathcal{A}, \mathcal{T}_\mathcal{A})$; in case of $\mathit{role\_expr} = \neg\mathtt{equal}$ we have $i^\mathcal{I} \neq j^\mathcal{I}$ in all models. A *predicate* (P) can be denoted as a lambda expression and is made from the vocabulary which Racer offers for building linear (in)equalities. For example, `(< (+ 20 age-1) age-2)` can be expressed as $P = \lambda(\mathit{age}_1, \mathit{age}_2).\mathit{age}_1 + 20 < \mathit{age}_2$:

$$
\begin{aligned}
&\mathsf{concept\_instances}(\mathcal{A}, \mathit{concept\_expr}) =_{def}\\
&\qquad \{\, i \mid i \in \mathsf{Inds}_\mathcal{A}, (\mathcal{A}, \mathcal{T}_\mathcal{A}) \models \mathit{concept\_expr}(i) \,\}\\
&\mathsf{role\_pairs}(\mathcal{A}, \mathit{role\_expr}) =_{def}\\
&\qquad \{\, \langle i, j \rangle \mid i, j \in \mathsf{Inds}_\mathcal{A}, (\mathcal{A}, \mathcal{T}_\mathcal{A}) \models \mathit{role\_expr}(i, j) \,\}\\
&\mathsf{predicate\_pairs}(\mathcal{A}, \mathit{attrib}_1, \mathit{attrib}_2, P) =_{def}\\
&\qquad \{\, \langle i, j \rangle \mid i, j \in \mathsf{Inds}_\mathcal{A}, (\mathcal{A}, \mathcal{T}_\mathcal{A}) \models\\
&\qquad\qquad \exists x, y : \mathit{attrib}_1(i, x) \wedge \mathit{attrib}_2(j, y) \wedge P(x, y) \,\}
\end{aligned}
$$

The semantics of complex nRQL bodies can be defined easily now:

$$\begin{aligned}
\texttt{(and } q_1 \ldots q_i\texttt{)}^{\mathcal{E}} &=_{def} \bigcap_{1\le j\le i} q_j^{\mathcal{E}} \\
\texttt{(union } q_1 \ldots q_i\texttt{)}^{\mathcal{E}} &=_{def} \bigcup_{1\le j\le i} q_j^{\mathcal{E}} \\
\texttt{(neg } q\texttt{)}^{\mathcal{E}} &=_{def} (\mathsf{Inds}_{\mathcal{A}})^n \setminus q^{\mathcal{E}} \\
\texttt{(project-to } (x_{i_1,q} \ldots x_{i_k,q})\ q\texttt{)}^{\mathcal{E}} &=_{def} \pi_{\langle i_1,\ldots,i_k\rangle}(q^{\mathcal{E}})
\end{aligned}$$

So far we have specified the semantics of a query body. To get the answer of a query, the *head* has to be considered. This can be seen as a further projection of $q^{\mathcal{E}}$ to the variables mentioned in the head. If the head contained an individual, then this individual has also been replaced by the representative variable in the head (see above). In case a head projection operator is encountered, the functional operator is applied to the binding of the argument variable. The value is included in the query answer (producing nested binding lists). In case of OWL datatype fillers, the projection operators will possibly return a list of datatype values. A formal definition of this process is omitted here due to space limitations.

△

Grounded conjunctive queries are a subset of the queries introduced above in which only conjunction (`and`) is used as an operator for query atoms. Note that they have to be distinguished from (full) conjunctive queries. The difference can be seen with the following example. Given the following TBox :

$$\begin{gathered}
Manager \doteq AreaManager \sqcup TopManager \\
AreaManager \sqsubseteq TopManager
\end{gathered}$$

and the following ABox:

$$\begin{gathered}
Manager(Andrea),\ TopManager(Mary),\ AreaManager(Paul), \\
Supervises(John, Andrea),\ Supervises(John, Mary), \\
OfficeMate(Mary, Andrea),\ OfficeMate(Andrea, Paul)
\end{gathered}$$

Let us assume that the query below is posed:

```
(retrieve (?x)(?x
               (some Supervises
                  (and TopManager (some OfficeMate AreaManager)))))
```

The result set of this query is $\{John\}$.

Although the only information given about *Andrea* is that *Andrea* is a *Manager*, from the given TBox it is clear that *Andrea* is either a *TopManager* or an *AreaManager*. In both cases, *John* is an instance of the query concept, and hence, *John* is a binding for $?x$. However, *John* is not in the result set of the following query (actually, the result set is empty):

```
(retrieve (?x) (and (?x ?y Supervises)
                    (?y TopManager)
                    (?y ?z OfficeMate)
                    (?z AreaManager)))
```

The reason is that *Andrea* is neither a binding for ?y nor a binding for ?z because it cannot be proved that *Andrea* is an instance of $TopManager$ nor can it be proved that *Andrea* is an $AreaManager$, respectively. *Andrea* is not a $TopManager$ (or $AreaManager$) in all models. Thus, grounded conjunctive queries have a different semantics as (full) conjunctive queries, which will be discussed below. Before we discuss conjunctive queries in detail, some comments on query engines are appropriate.

1.1.3 The nRQL Engine

The nRQL query answering engine implements the nRQL language as part of RacerPro 1.9. The engine offers *different querying modes:* basically, a synchronous *set-at-a-time mode* and an asynchronous *tuple-at-a-time mode.* In the *set-at-a-time mode*, a call to a querying function such as `retrieve` works synchronously. The client has to wait, the whole answer set is delivered in a bunch. However, many client applications prefer an asynchronous API, the *tuple-at-a-time mode* allows for incrementally loading the answer *tuple by tuple.* Thus, a call to `retrieve` will return immediately with a so-called *query identifier.* This identifier, say `:query-123`, can then be used as argument to functions such as `(get-next-tuple :query-123)`. Functions like `(get-next-n-tuples 10 :query-123)` are also provided. A similar way of client-server interaction is also presupposed in the OWL-QL query answering dialog. Thus, nRQL supports the OWL-QL query answering dialog quite well in this respect.

The incremental tuple-at-a-time mode can be used either *lazy* or *eager*. In the lazy mode, the next tuples will not be computed before requested by the client, unlike the eager mode, which pre-computes the next tuple(s) and puts them into a queue for future requests. In principle, there can be more than one running query at a time. The nRQL engine allows for concurrent querying. When a query is executed, a *thread* from a *pool of threads* is acquired and put to work. The engine can process up to a few hundred queries simultaneously, and serializes and minimizes the calls to the basic Racer ABox retrieval functions (e.g., `concept-instances`), by using locking techniques and dedicated index structures. Nearly all answers from Racer are cached, but the index structures are automatically invalidated if changes to the ABox (or TBox) occur. If a KB changes while queries are still active, then nRQL can be advised to deliver a *KB-has-changed-warning token* to the clients.

The *degree of completeness of query answering in nRQL is configurable:* If ABoxes get very big, it becomes impossible to use Racer's ABox retrieval functions for query answering. Even the required ABox consistency check will fail. Thus, nRQL can no longer be used in its complete mode. The incomplete modes can help, if "complete enough" for the application. These modes can still achieve a great deal of completeness. For example, using the incomplete nRQL mode "2", we observed in [43] that nRQL is still more complete than "DLDB" on the LUBM and achieves a comparable performance, even for "ABoxes" with a few 100.000 individuals.

Having incomplete modes available gives nRQL the ability to distinguish between cheap and expensive tuples. It is possible to advise nRQL first to deliver a set of *cheap tuples*, yielding an incomplete answer ("phase one"), and then to turn on Racers expensive ABox retrieval functions to deliver the remaining *expensive tuples* ("phase two"). We talk of *two-phase query processing modes.* Again, nRQL can be advised to deliver a *warning token* before phase two starts, informing the client that computation of the remaining tuples will possibly be expensive. The client can then chose to retrieve these additional tuples or not.

The engine supports *full life-cycle management for queries.* Queries can be prepared, made active, be suspended or aborted, eventually terminate, can be resurrected, etc. Runtime resources are configurable (size of thread pool, maximum bound on the number of answer tuples setable, timeout setable, tuple permutations can be excluded, etc.).

Another important feature of the engine is the *built-in query optimizer.* The basic idea is to reorder the query atoms in a conjunctive query in such a way as to heuristically minimize the number of *generators* required to compute candidate bindings for the variables. For example, for the query `(and (?x ?y R) (?y D) (?x C))`, the execution plan `(?x c), (?x ?y R), (?y D)` is preferable over the plan `(?x C), (?y D), (?x ?y R)`, since the second plan has to compute a Cartesian product, whereas the first plan can, once a binding for `?x` has been established, simply enumerate the `R` successors of `?x` for `?y` candidate generation, which is much more "local". In order to decide whether `(?y D), (?y ?x (inv R)), (?x C)` might even be better, nRQL uses ABox statistics and information from previously evaluated queries in order to implement the "most constrained generator first" heuristics.

1.1.4 Meta Level Querying

nRQL can also be used to search for specific superclass-class-subclass relationship patterns in the taxonomy of a TBox. Suppose that we view the taxonomy of a TBox as a relational structure. The taxonomy of a TBox is a so-called *directed acyclic graph (DAG).* A node in this DAG represents an *equivalence class* of *equivalent concept names*, and an edge between two nodes represents a *direct-subsumer-of relationship.*

Also assume that the following laws hold:

1. Assume that each node x has a name. The *name of the node* is the name of the equivalence class $[x]$. However, the name of this node might be any element from the equivalence class $[x]$.
2. For each node x (representing an equivalence class $[x]$) and each member $x_i \in [x]$ in this equivalence class, assume that the predicate x_i holds - thus, $x_i(x)$ is true. Since $x \in [x]$, also $x(x)$ holds.
3. The edges in this taxonomy representing the direct-subsumer-of relationship are labeled with *has_child*; that is, *has_child*(x, y) holds iff x is a *direct subsumer* of y.
4. Let *has_parent* be the inverse relationship of *has_child*, and *has_descendant* be the transitive closure of *has_child*. Let *has_ancestor* be the inverse relationship of *has_descendant*.

Obviously, such a relational structure can also be seen as an ABox. A *has_child*(x, y) edge is represented with a role membership axiom `(related x y has-child)`, and for each x, the concept membership axiom `(instance x x)` is added, and if `y` is in `(concept-synonyms x)`, then also `(instance x y)` is added, for all such `y`. But please note that the name of the node x might be any element from the equivalence class of x. We will call an ABox which is constructed according to the four laws the "taxonomy ABox". Note that this taxonomy ABox is not a real ABox.

It is obvious that we can now query this very specific taxonomy ABox with nRQL - thus, the full power of nRQL can be used for TBox querying purposes. However, set of roles that can be used in role query atoms is then limited to `has-child, has-parent, has-descendant, has-ancestor`, and the the set of concept expressions available in concept query atoms is limited to the set of concept names from the TBox (taxonomy). Of course, querying for concrete domain attributes etc. does not make sense in this setting, since the "taxonomy ABox" contains only information according to the given four rules above.

1.1.5 Access to Told Information

Commonly held view on ontology systems as a source of the inferred knowledge does not exclude the fact, that also the access to explicitly given information (so-called told information) is required for certain reasoning tasks. The access to told axioms can be implemented on the client-side, but then the client application has to keep and to maintain its own copy of the statements of the ontology. Obviously, a better solution would be to let the ontology server to manage told, pre-processed and inferred information in such a way that clients (humans or programs) can pose queries about told information.

As is known, modern DL reasoners implement various optimization techniques to achieve better performance. When processing ontologies, they apply a range of pre-processing algorithms (such as normalization and absorption), which syntacticly manipulate and simplify incoming KB axioms. Again, in order to provide better performance, only transformed axioms are kept for further processing. As a result of this preprocessing, building such server-side told information interface is far from being a trivial task.

1.1.6 Load Balancing and Caching

In our work we consider applications which generate queries w.r.t. many different knowledge bases. We presuppose that for a particular KB there exists many possible query servers. In order to successfully build applications that exploit these KB servers, an appropriate middleware is required. In particular, if there are many servers for a specific KB, the middleware is responsible for managing request dispatching and load balancing. Load balancing must be accompanied by middleware-side caching in order to reduce network latency.

In our view the KB servers we consider are managed by different organizations. Therefore, DL applications used in some company need some gateway inference server that provides local caching (in the intranet) to: (i) reduce external communication and (ii) avoid repetitive external server access operations in case multiple intranet applications pose the same queries.

In our case study we investigate a server for answering OWL-QL$^-$ queries[1]. This server

[1]OWL-QL$^-$ stands for OWL-QL with distinguished variables only.

(called RacerManager) acts as a proxy that delegates queries to back-end DL reasoners (RacerPro servers) that manage the KB mentioned in the query and load KBs on demand. Compared to previous versions, the functionality of RacerManager has been substantially enhanced. We address the problems of load balancing and caching strategies in order to exploit previous query results (possibly produced by different users of the local site). Caching is investigated in the presence of incrementally answered OWL-QL$^-$ queries. In addition, the effects of concurrent query executions on multiple (external) inference servers and corresponding transmissions of multiple partial result sets for queries are studied.

Reasoning over ontologies with a large number of individuals in ABoxes is a big challenge for existing reasoners. As discussed in Section 1.1.3, to deal with this problem, RacerPro supports iterative query answering, where clients may request partial result sets in the form of tuples. For iterative query answering, RacerPro can be configured to compute the next tuples on demand (lazy mode). Moreover, it can be instructed to return cheap (easily inferable) tuples first.

Although these configuration options enable the reasoner to achieve significant performance improvements for a single client, this effect decreases in scenarios where multiple clients pose queries concurrently. In fact, a single RacerPro instance cannot process several client requests in parallel. Thus, as long as RacerPro is processing a clients request, which usually includes activities such as parsing the query, reading the corresponding knowledge base, classifying it, finding requested number of answer tuples and returning them, all other clients have to wait in a queue.

Motivated by the concurrency problem, our OWL-QL$^-$ server is implemented to act as a load-balancing middleware between clients and multiple RacerPro instances. We chose a common n-tier architecture as the base layout. RacerManager can initialize and manage an array of RacerPro instances. Multiple clients can use the web service offered by RacerManager to send their OWL-QL$^-$ queries concurrently. With respect to the states of the managed RacerPro instances and a naive load-balancing strategy (similar to round-robin), RacerManager dispatches the queries to RacerPro instances. More precisely, given a query, which requires some ontology, RacerManager prefers RacerPro instances, which already worked on this ontology. Before a OWL-QL$^-$ query is send to a reasoner instance, it is translated to nRQL by the translator module. Preliminary test results showed that the proposed architecture prevents clients from blocking each other, as it is the case if multiple clients interact with a single reasoner.

Additionally, irrespective of load balancing and query dispatching, a client may benefit from the caching mechanism offered by RacerManager. In case he sends a query, which has been posed before, answer tuples are delivered directly from the cache. If the client requires more tuples than available in the cache, only the missing number of tuples is requested from an appropriate RacerPro instance. The cache expiration can be set to an arbitrary duration or turned off. In the latter case, the cache will never be cleared.

1.1.7 Related Work and Future Challenges

ABox retrieval languages are not new features in description logic inference systems. In particular, the design of nRQL is influenced by the query language of LOOM [66]. However, unlike the LOOM query language, nRQL is specified in a formally rigorous way.

For instance, the semantics offers projection operators in order to formalize predicates such as `has-known-successor`, which have been used in practical applications for a long time. In addition, the semantics for the ABox query language covers other facilities such as a negation as failure operator, which is also often used in application projects. With projection operators for concrete domain values, we need not impose complex safeness conditions.

Basic conjunctive query languages for description logics less expressive than the concept language of RacerPro (or OWL-DL without nominals) are formally investigated in [53]. There, so-called distinguished as well as non-distinguished variables are introduced. The distinguished variables correspond to nRQL's variables. The non-distinguished variables are purely existential variables. For example, if `!y` denotes such a non-distinguished variable, then the query `(and (?x c) (?x !y R))` is basically equivalent to `(?x (and c (some r top)))`. This observation also suggests a technique for "removing" non-distinguished variables, the so-called "rolling up" technique. However, this procedure only works for acyclic queries, and becomes more technically involved if more than one non-distinguished variable is present in the query. These theoretical techniques have been used as a basis for the implementation of an experimental DQL (DAML Query Language) server [35]. DQL is the predecessor of OWL-QL, and quite similar in many respects. However, this server can only handle acyclic conjunctive queries and does not offer negation as failure.

In the TONES project, optimization techniques for practical answering grounded conjunctive queries have been developed and published in [70]. While nRQL has been used in many application projects for some time now, based on [41] and [70] additional optimization techniques for implementing grounded conjunctive queries have been investigated with the Pellet description logic system [83].

In the future, new standards for query languages have to be developed such that different systems can be compared appropriately. In addition, important issues such as incremental changes to ABoxes as well as persistency have to be dealt with in order to support ontology-based information systems as a backbone of practical applications. Some reasoners already support retrieval of told information in some restricted form. E.g., in nRQL, concrete domain fillers and OWL annotations can be retrieved. But in general, a standard mechanism for retrieving and retracting the stored information as been told before is missing yet. This is the subject of recent efforts of the DIG 2.0 working group [63].

1.2 Expressive DLs and Conjunctive Queries

Standard DL reasoning services include testing concepts for satisfiability or retrieving instances of a given concept. The latter retrieves all (ABox) individuals that are an instance of the given (possibly complex) concept expression in every model of the knowledge base. The underlying reasoning problems are well-understood, and it is known that the combined complexity of these reasoning problems is ExpTime-complete for $\mathcal{SHIQ}$ [84], where $\mathcal{SHIQ}$ is the DL underlying DAML+OIL and OWL Lite. Despite this high worst-case complexity, efficient implementations of decision procedures for these problems are known. Furthermore, the TBox is usually small compared to the amount of data in the

ABox. Therefore, the data complexity of a reasoning problem, i.e., where the complexity is measured in the size of the ABox only, is often a more useful performance estimate. For $\mathcal{SHIQ}$, instance retrieval is known to be data complete for co-NP [55]. However, since instance retrieval only allows for querying the relational structure of the knowledge base in a restricted, tree-like way, it is commonly agreed that a more expressive query language is required, and that conjunctive queries are a suitable basis for this.

In this document, we discuss conjunctive query answering in $\mathcal{SHIQ}$: the presence of transitive and inverse roles makes the problem rather tricky [36], and results are only available for two kinds of restrictions. The first kind, *grounded* conjunctive queries, is obtained by restricting the variables in queries to be bound to individual names in the ABox only. This results in a form of closed-domain semantics which is different from the usual open-world (and open-domain) semantics in DLs. Motik et al. [71] show that answering *grounded* conjunctive queries for $\mathcal{SHIQ}$ is decidable, and they form the basis for the query language nRQL [42]. In the second kind, the binary atoms in conjunctive queries are restricted to *simple* roles, i.e., to those that are neither transitive nor have transitive sub-roles. For this restriction, decision procedures for various DLs around $\mathcal{SHIQ}$ are known [52, 73], and it is known that answering conjunctive queries is data complete for co-NP [73].

We present a decision procedure for conjunctive query answering over $\mathcal{SHIQ}$ knowledge bases without any of these restrictions. We achieve this by transforming the conjunctive query into $\mathcal{SHIQ}^{\sqcap}$-concepts,[2] and showing that conjunctive query answering can be reduced to consistency of $\mathcal{SHIQ}$-knowledge bases extended with $\mathcal{SHIQ}^{\sqcap}$ assertions and GCIs. From our decision procedure, it follows that conjunctive query entailment is data complete for co-NP, and can be decided in time double exponential in the size of the query and single exponential in the size of the knowledge base.

We first introduce the syntax and semantics of $\mathcal{SHIQ}$ and conjunctive queries.

1.2.1 Syntax and Semantics of $\mathcal{SHIQ}$

Let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ be sets of *concept names*, *role names*, and *individual names*. We assume that the set $\mathsf{N_R}$ or role names is partitioned into a set $\mathsf{N_{tR}}$ of *transitive role names* and a set $\mathsf{N_{rR}}$ of *normal role names*, i.e., $\mathsf{N_{tR}} \cup \mathsf{N_{rR}} = \mathsf{N_R}$ with $\mathsf{N_{tR}} \cap \mathsf{N_{rR}} = \emptyset$. A *role* is an element of $\mathsf{N_R} \cup \{r^- \mid r \in \mathsf{N_R}\}$, where roles of the form r^- are called *inverse roles*. A *role inclusion* is of the form $r \sqsubseteq s$ with r and s as roles. A *role hierarchy* $\mathcal{H}$ is a finite set of role inclusions.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, which maps every concept name A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role name $r \in \mathsf{N_{rR}}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, every role name $r \in \mathsf{N_{tR}}$ to a transitive binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a role inclusion $r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ and a role hierarchy $\mathcal{H}$ if it satisfies all role inclusions in $\mathcal{H}$. We use the following standard notation:

1. We define a function Inv which returns the inverse of a role. More precisely, $\mathsf{Inv}(r) := r^-$ if $r \in \mathsf{N_R}$ and $\mathsf{Inv}(r) := s$ if $s = r^-$ for a role name s.

[2] $\mathcal{SHIQ}^{\sqcap}$ is $\mathcal{SHIQ}$ plus role conjunction.

2. Since set inclusion is transitive, we define, for a role hierarchy $\mathcal{H}$, $\sqsubseteq^*_{\mathcal{H}}$ as the reflexive transitive closure of $\sqsubseteq$ over $\mathcal{H} \cup \{\mathsf{Inv}(r) \sqsubseteq \mathsf{Inv}(s) \mid r \sqsubseteq s \in \mathcal{H}\}$. We use $r \equiv^*_{\mathcal{H}} s$ as an abbreviation for $r \sqsubseteq^*_{\mathcal{H}} s$ and $s \sqsubseteq^*_{\mathcal{H}} r$.

3. For a role hierarchy $\mathcal{H}$ and a role s, we define the set $\mathsf{Trans}_{\mathcal{H}}$ of transitive roles as $\{s \mid$ there is a role r with $r \equiv s$ and $r \in \mathsf{N_{tR}}$ or $\mathsf{Inv}(r) \in \mathsf{N_{tR}}\}$.

4. A role r is called *simple* w.r.t. a role hierarchy $\mathcal{H}$ if for each role s such that $s \sqsubseteq^*_{\mathcal{H}} r$, $s \notin \mathsf{Trans}_{\mathcal{H}}$.

The subscript $\mathcal{H}$ of $\sqsubseteq^*_{\mathcal{H}}$ and $\mathsf{Trans}_{\mathcal{H}}$ is dropped if clear from the context.

The set of $\mathcal{SHIQ}$*-concepts* (or concepts for short) is the smallest set built inductively from $\mathsf{N_C}$ using the following grammar, where $A \in \mathsf{N_C}$, $n \in \mathbb{N}$, C, C_1, C_2 are concepts, r is a role and s is a simple role:

$$C ::= \top \mid \bot \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall r.C \mid \exists r.C \mid \leqslant ns.C \mid \geqslant ns.C.$$

The semantics of $\mathcal{SHIQ}$-concepts is defined as follows:

$$\begin{array}{rl}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\bot^{\mathcal{I}} = \emptyset & (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall r.C)^{\mathcal{I}} = & \{d \in \Delta^{\mathcal{I}} \mid \text{ if } (d, d') \in r^{\mathcal{I}}, \text{ then } d' \in C^{\mathcal{I}}\} \\
(\exists r.C)^{\mathcal{I}} = & \{d \in \Delta^{\mathcal{I}} \mid \text{ There is a } (d, d') \in r^{\mathcal{I}} \text{ with } d' \in C^{\mathcal{I}}\} \\
(\leqslant ns.C)^{\mathcal{I}} = & \{d \in \Delta^{\mathcal{I}} \mid |s^{\mathcal{I}}(d, C)| \leqslant n\} \\
(\geqslant ns.C)^{\mathcal{I}} = & \{d \in \Delta^{\mathcal{I}} \mid |s^{\mathcal{I}}(d, C)| \geqslant n\}
\end{array}$$

where $|M|$ denotes the cardinality of the set M and $s^{\mathcal{I}}(d, C)$ is defined as

$$\{d' \in \Delta^{\mathcal{I}} \mid (d, d') \in s^{\mathcal{I}} \text{ and } d' \in C^{\mathcal{I}}\}.$$

A *general concept inclusion* (GCI) is an expression $C \sqsubseteq D$, where both C and D are concepts. A finite set of GCIs is called a *TBox*. An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and a TBox $\mathcal{T}$ if it satisfies each GCI in $\mathcal{T}$. An *assertion* is an expression of the form $C(a)$, $r(a, b)$, $\neg r(a, b)$, or $a \neq b$, where C is a concept, r is a role, $a, b \in \mathsf{N_I}$. An *ABox* is a finite set of assertions. We use $\mathsf{Ind}(\mathcal{A})$ to denote the set of individual names occurring in $\mathcal{A}$, and if $\mathcal{A}$ is clear from the context, we write only Ind. An interpretation $\mathcal{I}$ *satisfies* an assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, $\neg r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$, and $a \neq b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* an ABox if it satisfies each assertion in $\mathcal{A}$, which we denote with $\mathcal{I} \models \mathcal{A}$.

A *knowledge base* (KB) is a triple $(\mathcal{T}, \mathcal{H}, \mathcal{A})$ with $\mathcal{T}$ a TBox, $\mathcal{H}$ a role hierarchy, and $\mathcal{A}$ an ABox. Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a KB and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ an interpretation. We say that $\mathcal{I}$ *satisfies* $\mathcal{K}$ if $\mathcal{I}$ satisfies $\mathcal{T}$, $\mathcal{H}$, and $\mathcal{A}$. In this case, we say that $\mathcal{I}$ is a *model* of $\mathcal{K}$ and write $\mathcal{I} \models \mathcal{K}$. We say that $\mathcal{K}$ *is consistent* if $\mathcal{K}$ has a model.

1.2.2 Conjunctive Queries Tasks

Now that we have defined the syntax and semantics of $\mathcal{SHIQ}$-concepts and knowledge bases, we are ready to introduce conjunctive queries and the *reasoning tasks* regarding conjunctive queries. Let $\mathsf{N_V}$ be a countably infinite set of *variables* disjoint from $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$. Moreover, let $\mathsf{N_P} = \mathsf{N_C} \cup \mathsf{N_R}$ be the set of *predicate names*.

An *atom* is an expression $A(v)$ or $r(v, v')$, where $A \in \mathsf{N_C}$, r is a role, and $v, v' \in \mathsf{N_V}$. A *conjunctive query* q is a non-empty set of atoms. Intuitively, such a set represents the conjunction of the atoms in the set. We use $\mathsf{Var}(q)$ to denote *the set of variables occurring in* q and we define the *size* $|q|$ of q as the number of atoms in q. Let $\mathcal{I}$ be an interpretation, q a conjunctive query, and $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ a total function. We write

- $\mathcal{I} \models^{\pi} A(v)$ if $(\pi(v)) \in A^{\mathcal{I}}$;
- $\mathcal{I} \models^{\pi} r(v, v')$ if $(\pi(v), \pi(v')) \in r^{\mathcal{I}}$;

If $\mathcal{I} \models^{\pi} at$ for all $at \in q$, we write $\mathcal{I} \models^{\pi} q$. We say that $\mathcal{I}$ *satisfies* q and write $\mathcal{I} \models q$ if there is a π with $\mathcal{I} \models^{\pi} q$.

Task 1: Query Answering One reasoning task regarding conjunctive queries is *query answering*. For introducing this reasoning task, let the variables of a conjunctive query be typed: each variable can either be *non-distinguished*, i.e., existentially quantified or *distinguished*. We call distinguished variables also *answer variables*. Let q be a query in n variables, of which $v_1, \ldots, v_m$ $(m \leq n)$ are distinguished. The *answers* of $\mathcal{K}$ to q are those m-tuples $(a_1, \ldots, a_m) \in \mathsf{N_I}^m$ such that for all models $\mathcal{I}$ of $\mathcal{K}$, $\mathcal{I} \models^{\pi} q$ for some π that satisfies $\pi(v_i) = a_i$ for all i with $1 \leq i \leq m$. Observe that we admit only concept names in atoms $A(v)$, but no complex concepts. This is no restriction since an atom $C(a)$ with C complex can be simulated using the atom $A(a)$ and the concept inclusion $C \sqsubseteq A$.

Task 2: Query Containment Query containment is important in many areas, including information integration, query optimization, and reasoning about Entity-Relationship diagrams. A query q is contained in a query q' w.r.t. a knowledge base $\mathcal{K}$, if $q^{\mathcal{I}} \subseteq q'^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$. Although query containment is not the same as query answering, it can be used to answer boolean queries by encoding the ABox into the less general query, i.e., q in the given example. If the query q' is contained in the encoded ABox, the query answer for q' is true. This technique is well known in the database community.

Task 3: Query Entailment A reasoning task closely related to query answering is *query entailment*. Here we are given a knowledge base $\mathcal{K}$ and query q and asked whether $\mathcal{I} \models q$ for all models $\mathcal{I}$ of $\mathcal{K}$. If this is the case, we say that $\mathcal{K}$ *entails* q and write $\mathcal{K} \models q$.

In this document, we focus on query entailment. The reasons for this are two-fold: first, query answering can be reduced to query entailment. And second, in contrast to query answering, query entailment is a decision problem and can be studied in terms of classical complexity theory.

We now make the connection between query answering and query entailment more precise. Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base, q a conjunctive query in n variables with answer variables $v_1, \ldots, v_m$, and $t = (a_1, \ldots, a_m) \in \mathsf{N_I}^m$ a tuple. Our aim is to reduce checking whether t is an answer of $\mathcal{K}$ to a to query entailment. To this end, let $\mathcal{A}' := \mathcal{A} \cup \{A_i(a_i) \mid 1 \leq i \leq m\}$, where $A_1, \ldots, A_m$ are concept names that do not occur in $\mathcal{K}$. Moreover, let $\mathcal{K}' := (\mathcal{T}, \mathcal{H}, \mathcal{A}')$ and let $q' := q \cup \{A_i(v_i) \mid 1 \leq i \leq m\}$. The following is not difficult to prove.

Lemma 3. *The tuple t is an answer of $\mathcal{K}$ to q iff $\mathcal{K}'$ entails q'.*

This technique is well known and the newly introduced concept names are often referred to as representative concepts [52] or name formulae [19]. The same technique can be used in order to represent constants (individual names) in the query.

In what follows, we assume for convenience that conjunctive queries are closed under inverses, i.e., if $r(v, v') \in q$, then $\mathsf{Inv}(r)(v', v) \in q$ and if we add or remove atoms from a query, we implicitly assume that we do this such that the resulting query is again closed under inverses. We will also assume that queries are connected. More precisely, let q be a conjunctive query. We say that q is *connected* if for all $v, v' \in \mathsf{Var}(q)$, there exists a sequence $v_0, \ldots, v_{n-1}$ such that $v_0 = v$, $v_{n-1} = v'$, and for all $i < n - 1$, there exists a role r such that $r(v_i, v_{i+1}) \in q$. A collection $q_1, \ldots, q_k$ of queries is a *partitioning* of q if $q = q_1 \cup \cdots \cup q_k$, $\mathsf{Var}(q_i) \cap \mathsf{Var}(q_j) = \emptyset$ for $1 \leq i < j \leq k$, and each q_i is connected. The next lemma says that we can restrict ourselves to the entailment of connected queries. In what follows, we assume queries to be connected without further notice.

Lemma 4. *Let $\mathcal{K}$ be a knowledge base, q a conjunctive query, and $q_1, \ldots, q_n$ a partitioning of q. Then $\mathcal{K} \models q$ iff $\mathcal{K} \models q_i$ for $1 \leq i \leq n$.*

In what follows, we use q for a connected conjunctive query and $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ for a knowledge base such that, in all assertions $C(a) \in \mathcal{A}$, C is a (possibly negated) concept name. Moreover, for a mapping f, we use $\mathsf{dom}(f)$ and $\mathsf{ran}(f)$ to denote f's domain and range, respectively.

1.2.3 Forests and Trees

We will first define canonical (forest-shaped) interpretations, and prove that we can limit our attention to such interpretations.

Definition 5. Let $\mathbb{N}^*$ be the set of all (finite) words over the alphabet $\mathbb{N}$. A *tree* T is a non-empty prefix-closed subset of $\mathbb{N}^*$. For $w, w' \in T$, we call w' a *successor* of w if $w' = w \cdot c$ for some $c \in \mathbb{N}$, where "$\cdot$" denotes concatenation. We call w' a *neighbor* of w if w' is a successor of w or vice versa. Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a $\mathcal{SHIQ}$ knowledge base. A *forest base for* $\mathcal{K}$ is an interpretation $\mathcal{I}$ that interpretes transitive roles in unrestricted (i.e., not necessarily transitive) relations and additionally satisfies the following conditions:

T1 $\Delta^\mathcal{I} \subseteq \mathsf{Ind}(\mathcal{A}) \times \mathbb{N}^*$ such that for all $a \in \mathsf{Ind}(\mathcal{A})$, the set $\{w \mid (a, w) \in \Delta^\mathcal{I}\}$ is a tree;

T2 if $((a, w), (a', w')) \in r^\mathcal{I}$, then either $w = w' = \varepsilon$ or $a = a'$ and w' is a neighbor of w;

T3 for all $a \in \mathsf{Ind}(\mathcal{A})$, $a^\mathcal{I} = (a, \varepsilon)$.

Let $\mathcal{I}$ be an interpretation. Then $\mathcal{I}$ is *canonical for* $\mathcal{K}$ if there exists a forest base $\mathcal{J}$ for $\mathcal{K}$ such that $\mathcal{J}$ is identical to $\mathcal{I}$ except that, for all non-simple roles r, we have

$$r^\mathcal{I} = r^\mathcal{J} \cup \bigcup_{s \sqsubseteq^* r,\, s \in \mathsf{Trans}} (s^\mathcal{J})^+$$

In this case, we say that $\mathcal{J}$ is a forest base *for* $\mathcal{I}$. △

Observe that if $\mathcal{I}$ is canonical for $\mathcal{K}$, then $\Delta^{\mathcal{I}}$ satisfies Condition $\mathsf{T1}$ and $\mathsf{T3}$ above.

Lemma 6. *$\mathcal{K} \not\models q$ iff there exists a canonical model $\mathcal{I}$ with $\mathcal{I} \not\models q$.*

In order to decide whether $\mathcal{K} \models q$, our algorithm will check for the existence of a *counter model*, i.e., a model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models q$. Obviously, the above observation means that it suffices to look only for canonical counter models.

Let $\mathcal{I}$ be a canonical model of $\mathcal{K}$, and $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ such that $\mathcal{I} \models^{\pi} q$. We say that π is a *forest match* if for all $r(v, v') \in q'$, we have one of the following:

- $\pi(v) = (a, \varepsilon)$ and $\pi(v') = (b, \varepsilon)$ for some $a, b \in \mathsf{Ind}(\mathcal{A})$;
- $\pi(v) = (a, w)$ and $\pi(v') = (a, w')$ for some $a \in \mathsf{Ind}(\mathcal{A})$ and $w, w' \in \mathbb{N}^*$.

Let $\mathcal{I}$ be a canonical model and π a forest match. A variable v is *grounded w.r.t. $\mathcal{I}$ and π* if $\pi(v) = (a, \varepsilon)$ for some $a \in \mathsf{Ind}(\mathcal{A})$. A forest match π defines a "partial grounding" for q, and allows us to view q as being split into a set of sub-queries, each of which is mapped into a single tree of $\mathcal{I}$.

We will now describe a series of transformations that we will apply to a query. The first of these, transitive rewriting, will allow us to restrict our attention to forest matches.

Definition 7. Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base and q a conjunctive query. Then a query q' is called a *transitive rewriting* of q w.r.t. $\mathcal{K}$ (or simply a transitive rewriting when $\mathcal{K}$ is obvious from the context) if it is obtained from q by choosing atoms $r_0(v_0, v_0'), \ldots, r_n(v_n, v_n') \in q$ and roles $s_0, \ldots, s_n \in \mathsf{Trans}$ such that $s_i \sqsubseteq^* r_i$ for all $i \leq n$, and then replacing $r_i(v_i, v_i')$ with

$$s_i(v_i, u_i), s_i(u_i, u_i'), s_i(u_i', v_i')$$
$$\text{or}$$
$$s_i(v_i, u_i), s_i(u_i, v_i')$$

for all $i \leq n$, where u_i and u_i' are variables that do not occur in q. We use $\mathsf{tr}_{\mathcal{K}}(q)$ to denote the set of all transitive rewritings of q w.r.t. $\mathcal{K}$. △

We assume that $\mathsf{tr}_{\mathcal{K}}(q)$ contains no isomorphic queries, i.e., differences in (newly introduced) variable names only are neglected.

Together with Lemma 6, the following lemma shows that in order to decide whether $\mathcal{K}$ entails q, we may enumerate all transitive rewritings q' of q and check whether there is a canonical model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \models^{\pi} q$ with π a forest match.

Lemma 8. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base, q a conjunctive query, and $\mathcal{I}$ a model of $\mathcal{K}$. Then the following holds:*

1. *If $\mathcal{I}$ is canonical and $\mathcal{I} \models q$, then there is a $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ such that $\mathcal{I} \models^{\pi'} q'$, with π' a forest match.*
2. *If $\mathcal{I} \models q'$ with $q' \in \mathsf{tr}_{\mathcal{K}}(q)$, then $\mathcal{I} \models q$.*

Lemma 9. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base, q a query, $|q| = n$, and $|\mathcal{H}| = m_{\mathcal{H}}$. Then there is a polynomial p such that*

(a) $|\mathsf{tr}_{\mathcal{K}}(q)| \leq 2^{p(n)\cdot \log p(m_{\mathcal{H}})}$

(b) for all $q' \in \mathsf{tr}_{\mathcal{K}}(q)$, $|q'| \leq p(n)$,

Let q be a query and $\mathcal{I}$ a canonical interpretation. A special case of forest matches are *tree matches*, i.e., matches $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ for which there exists an $a_0 \in \mathsf{Ind}(\mathcal{A})$ such that for all $v \in \mathsf{Var}(q)$, we have $\pi(v) = (a_0, w)$ for some $w \in \mathbb{N}^*$. Intuitively, in this case the whole match concerns only one of the trees in the forest $\Delta^{\mathcal{I}}$, and we call π an *a-tree match* if, for each $v \in \mathsf{Var}(q)$, there is some w such that $\pi(v) = (a, w)$. In our algorithm, forest matches of a query q will be broken down into tree matches of subqueries of q.

We will now show how a query can be rewritten as a tree-shaped query. This procedure, which we call *tree transformation*, can be applied to the sub-queries identified by a forest match; we can then use rolling-up to transform each sub-query into a concept.

Tree transformation of q is a three stage process. In the first stage, we derive a *collapsing* q_0 of q by (possibly) identifying variables in q. This allows us, e.g., to transform atoms $r(v, u), r(v, u'), r(u, w), r(u', w)$ into a tree shape by identifying u and u'. In the second stage, we derive an *extension* q_1 of q_0 by (possibly) introducing new variables and role atoms that make redundant existing role atoms $r(v, v')$, where r is non-simple. In the third stage, we derive a *reduct* q' of q_1 by (possibly) removing redundant role atoms, i.e., atoms $r(v, v')$ such that there exist variables $v_0, \dots, v_n \in \mathsf{Var}(q_1)$ with $v_0 = v$, $v_n = v'$, $s(v_i, v_{i+1}) \in q_1$ for all $i < n$, $s \sqsubseteq^* r$, and $s \in \mathsf{Trans}$. Combining the extension and reduct steps allows us, e.g., to transform a "loop" $r(v, v)$ into a tree shape by introducing a new variable v' and edges $s(v, v'), s(v', v)$ such that $s \sqsubseteq^* r$ and $s \in \mathsf{Trans}$, and then removing the redundant atom $r(v, v)$.

We will now describe this procedure more formally.

Definition 10. Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base. A conjunctive query q is *tree-shaped* if there exists a bijection τ from $\mathsf{Var}(q)$ into a tree such that $r(v, v') \in q$ implies that $\tau(v)$ is a neighbor of $\tau(v')$. Then

- a *collapsing* of q is obtained by identifying variables in q.
- the query q' is an *extension* of q w.r.t. $\mathcal{K}$ if the following hold:
  1. $q \subseteq q'$;
  2. $A(v) \in q'$ implies $A(v) \in q$;
  3. $r(v, v') \in q' \setminus q$ implies that r occurs in $\mathcal{H}$;
  4. $|\mathsf{Var}(q')| \leq 4|q|$;
  5. $|\{r(v, v') \in q' \mid r(v, v') \notin q\}| \leq 171|q|^2$.
- the query q' is a *reduct* of q w.r.t. $\mathcal{K}$ if the following hold:
  1. $q' \subseteq q$;
  2. $A(v) \in q$ implies $A(v) \in q'$;

3. if $r(v, v') \in q \setminus q'$, then there is a role s such that $s \sqsubseteq^* r$, $s \in \mathsf{Trans}$, and there are $v_0, \ldots, v_n$ such that $v_0 = v$, $v_n = v'$, and $s(v_i, v_{i+1}) \in q'$ for all $i < n$.

- a *tree transformations* of q is a query q' for which there are queries q_0 and q_1 such that
  - q_0 is a collapsing of q;
  - q_1 is an extension of q_0 w.r.t. $\mathcal{K}$;
  - q' is a tree-shaped reduct of q_1.

We use $\mathsf{tt}_{\mathcal{K}}(q)$ to denote the set of all tree transformations of q w.r.t. $\mathcal{K}$. △

We note that Condition 5 of extensions is not strictly needed. However, without this condition the algorithm for query entailment to be developed would require double exponential time in the size of the input knowledge base instead of only single exponential time. As in the case of $\mathsf{tr}_{\mathcal{K}}(q)$, we assume that $\mathsf{tt}_{\mathcal{K}}(q)$ does not contain any isomorphic queries.

We now derive an upper bound on the number and size of elements in $\mathsf{tt}_{\mathcal{K}}(q)$. The *size* $|\mathcal{T}|$ ($|\mathcal{H}|, |\mathcal{A}|$) of $\mathcal{T}$ ($\mathcal{H}$, $\mathcal{A}$) is the number of symbols needed to write it. For a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$, the *size* $|\mathcal{K}|$ of $\mathcal{K}$ is the number of symbols needed to write all the components $\mathcal{T}$, $\mathcal{H}$, and $\mathcal{A}$ of $\mathcal{K}$.

Lemma 11. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base, q a query, $|q| = n$, and $|\mathcal{H}| = m_{\mathcal{H}}$. Then the following hold:*

(a) $|\mathsf{tt}_{\mathcal{K}}(q)| \leq 2^{p(n) \cdot \log p(m_{\mathcal{H}})}$

(b) for all $q' \in \mathsf{tt}_{\mathcal{K}}(q)$, $|q'| \leq p(n)$,

where p is a polynomial.

Let $\mathcal{K}$ be a knowledge base, q a query, and $q' \in \mathsf{tt}_{\mathcal{K}}(q)$. For each $v \in \mathsf{Var}(q)$, let $\sigma(v)$ be the variable in $\mathsf{Var}(q')$ that v has been identified with ($\sigma(v) = v$ if v has not been identified with another variable). Take mappings $\pi : \mathsf{Var}(q) \to \mathbb{N}^*$ and $\pi' : \mathsf{Var}(q') \to \mathbb{N}^*$. We call π and π' *ε-compatible* iff, for all variables $v \in \mathsf{Var}(q)$, $\pi(v) = \varepsilon$ iff $\pi'(\sigma(v)) = \varepsilon$. Since q' is tree-shaped, π' is a tree with ε as the root and intuitively, ε-compatibility then also guarantees us that we can use $v \in \mathsf{Var}(q)$ for which $\pi(v) = \varepsilon$ as the root or starting point in π and use the above defined transformations in order to transform q into q'.

Lemma 12. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ be a knowledge base, $\mathcal{I}$ a canonical model of $\mathcal{K}$, q a conjunctive query, and π an a-tree match. If $\mathcal{I} \models^{\pi} q$, then there is a $q' \in \mathsf{tt}_{\mathcal{K}}(q)$ and an a-tree match π' such that $\mathcal{I} \models^{\pi'} q'$ and π and π' are ε-compatible.*

Lemma 13. *Let $\mathcal{I}$ be an interpretation, q a query, $q' \in \mathsf{tt}_{\mathcal{K}}(q)$, and π' a mapping such that $\mathcal{I} \models^{\pi'} q'$. Then there is a π such that $\mathcal{I} \models^{\pi} q$ and π and π' are ε-compatible.*

Let q be a conjunctive query. It is easy to see how to produce the set S of all reducts of extensions of collapsings of q. To select the tree-shaped queries from S, we may proceed as follows. Let $q' \in S$ and select a $v_r \in \mathsf{Var}(q')$. Then define a mapping $\tau : \mathsf{Var}(q') \to \mathbb{N}^*$ inductively as follows:

- Initially, set $\tau(v_r) := \varepsilon$;
- if $\tau(v)$ is already defined and
$$V = \{v' \in \mathsf{Var}(q) \mid r(v, v') \text{ for some role } r \text{ and } \tau(v') \text{ undefined}\},$$
then fix an injection $f : V \to \mathbb{N}$ and set $\tau(v') = \tau(v) \cdot f(v')$ for all $v' \in V$.

Clearly, $\mathsf{ran}(\tau)$ is a tree. The following is not difficult to prove.

Lemma 14. *The query q' is tree-shaped iff for all $r(v, v') \in q'$, $\tau(v)$ is a neighbor of $\tau(v')$.*

The algorithm to be designed in the following section crucially relies on the observation that tree-shaped queries can be converted into concepts formulated in the description logic $\mathcal{ELI}^{\sqcap}$, which offers only the concept constructors $\sqcap$ and $\exists r_0 \sqcap \cdots \sqcap r_{n-1}.C$, where $r_0, \ldots, r_{n-1}$ are (possibly inverse or non-simple) roles. The semantics of the latter operator is as follows:

$$(\exists r_0 \sqcap \cdots \sqcap r_{n-1}.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \exists e : (d, e) \in r_i^{\mathcal{I}} \text{ for } 0 \leqslant i < n \text{ and } e \in C^{\mathcal{I}}\}.$$

More precisely, this conversion can be done as follows. Let q be a tree-shaped query and $\tau : \mathsf{Var}(q) \to \mathbb{N}^*$ with $\varepsilon \in \mathsf{ran}(\tau)$ such that $r(v, v') \in q$ iff $\tau(v)$ is a neighbor of $\tau(v')$. Then assign to each variable v a concept $C_q(v)$ by proceeding in a bottom-up fashion through the tree $\mathsf{ran}(\tau)$:

- if $\tau(v)$ is a leaf of $\mathsf{ran}(\tau)$, then $C_q(v) := \bigsqcap_{A(v) \in q} A$
- if $\tau(v)$ has successors $\tau(v_0), \ldots, \tau(v_{n-1})$, then
$$C_q(v) := \bigsqcap_{A(v)\in q} A \sqcap \bigsqcap_{0 \leqslant i < n} \exists \Big(\bigsqcap_{r(v, v_i) \in q} r \Big).C_q(v_i).$$

Then C_q is $C_q(v_r)$ for $\tau(v_r) = \varepsilon$.

Lemma 15. *Let q be a tree-shaped query, $\mathcal{I}$ an interpretation, and $v_r \in \mathsf{Var}(q)$. Then $\mathcal{I} \models q$ iff $C_q(v_r)^{\mathcal{I}} \neq \emptyset$. In particular, $d \in C_q(v_r)^{\mathcal{I}}$ implies that there is a π with $v_r \mapsto d$ such that $\mathcal{I} \models^{\pi} q$.*

Lemma 15 shows that for all queries q, interpretations $\mathcal{I}$, and variables $v, v' \in \mathsf{Var}(q)$, we have $C_q(v)^{\mathcal{I}} \neq \emptyset$ iff $C_q(v')^{\mathcal{I}} \neq \emptyset$. This justifies the following: given a conjunctive query q, we use C_q to denote $C_q(v)$ for some arbitrary (but fixed) $v \in \mathsf{Var}(q)$.

We could now apply tree transformations to the sub-queries identified by a forest match, and use so-called representative concepts [52] or name formulae [19] to roll up the resulting query into a concept C_q. This would allow us to straightforwardly obtain a decision procedure: $\mathcal{K} \models q$ iff for every model $\mathcal{I}$ of $\mathcal{K}$ there is some C such that C is a concept that can be obtained by rolling-up a tree transformation of the sub-queries identified by a forest match of a transitive rewriting of q, and $C^{\mathcal{I}} \neq \emptyset$. If $\mathbf{C}$ is the set of all such concepts, then for $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$, $\mathcal{K} \models q$ iff $(\mathcal{T}', \mathcal{H}, \mathcal{A})$ is inconsistent, where

$$\mathcal{T}' = \mathcal{T} \cup \{\top \sqsubseteq \neg C \mid C \in \mathbf{C}\}.$$

By doing so, however, we would compromise the clear separation between the TBox and the ABox, and thus we could no longer obtain tight data complexity results. We will therefore present a decision procedure that uses extended ABoxes to check for the existence of forest matches; this decision procedure yields the desired complexity results.

1.2.4 The Decision Procedure

In order to gain insight into the data complexity of query entailment, we devise a procedure that uses extensions of both TBox and ABox. We proceed as follows: roughly speaking, we look for a KB $\mathcal{K}'$ such that $\mathcal{K}'$ extends $\mathcal{K}$ (both w.r.t. TBox and Abox), and the additional axioms and assertions prevent the existence of a transitive rewriting q' of q, a canonical model $\mathcal{I}$ of $\mathcal{K}$, and a forest match π such that $\mathcal{I} \models^{\pi} q'$. Lemmas 6 and 8 and the fact that $\mathcal{K}'$ extends $\mathcal{K}$ clearly also implies $\mathcal{K} \not\models q$. We consider all "relevant" extensions of $\mathcal{K}$ so that, if we find no extension $\mathcal{K}'$ such that $\mathcal{K}' \not\models q$, we can conclude that $\mathcal{K} \models q$.

In order to define $\mathcal{K}'$, we use Lemma 15, and thus $\mathcal{K}'$ will not be a $\mathcal{SHIQ}$ knowledge base. An *extended knowledge base* $\mathcal{K}'$ is of the form $(\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$ with

- $\mathcal{T}$, $\mathcal{H}$, and $\mathcal{A}$ are as in a $\mathcal{SHIQ}$ knowledge base;
- $\mathcal{T}_q$ is a finite set of GCIs $\top \sqsubseteq C$ with C a $\mathcal{SHIQ}^{\sqcap}$ concept;
- $\mathcal{A}'$ is an ABox such that if $C(a) \in \mathcal{A}'$, then $a \in \mathsf{Ind}(\mathcal{A})$ and C is a negated $\mathcal{SHIQ}^{\sqcap}$ concept.

The extended knowledge bases $\mathcal{K}'$ that we construct from $\mathcal{K}$ and q will be such that every counter model against $\mathcal{K} \models q$ (i.e., $\mathcal{I} \not\models q$) is a model of some $\mathcal{K}'$ and, for each model $\mathcal{I}$ of a $\mathcal{K}'$, we have that $\mathcal{I} \not\models q$. Thus, $\mathcal{K} \models q$ iff each $\mathcal{K}'$ is inconsistent. From Lemmas 6 and 8, to ensure that models of the $\mathcal{K}'$ are counter models, it suffices to prevent forest matches of transitive rewritings of q w.r.t. $\mathcal{K}$ in canonical models of $\mathcal{K}'$—and this is the role played by $\mathcal{T}_q$ and $\mathcal{A}'$. We distinguish between two kinds of forest matches: a-tree matches and *true* forest matches, i.e., forest matches that are not a-tree matches. To prevent a-tree matches, it suffices to consider the tree transformations of q. Therefore, $\mathcal{T}_q$ is defined as follows:

$$\mathcal{T}_q = \{\top \sqsubseteq \neg C_{q'} \mid q' \in \mathsf{tt}_{\mathcal{K}}(q)\}.$$

To prevent true forest matches, we further include an ABox $\mathcal{A}'$, which contains additional information about the individuals in $\mathcal{A}$. This is similar to the well-known precompletion approach for reducing ABox consistency to concept satisfiability [48]. Each $\mathcal{A}'$ represents a model in which there is no "true forest match" of a transitive rewriting of q, i.e., it contains, for each possible forest match, an assertion that "spoils" it.

This can consist either of an assertion which ensures that, for some grounded variable $v \in \mathsf{Var}(q')$ with $\pi(v) = (a, \varepsilon)$, a is not an instance of any rolling-up of a tree transformation of the a-tree match containing v, or of an assertion that ensures, for some grounded variables $v, v' \in \mathsf{Var}(q')$, with $r(v, v') \in q'$, $\pi(v) = (a, \varepsilon)$ and $\pi(v') = (b, \varepsilon)$, a is not r-related to b (i.e., $\neg r(a, b)$).

In the following, a *sub-query* of q is simply a non-empty subset of q (including q itself). Let Q be the set of all queries that are a tree transformation of a sub-query of a transitive rewriting of q w.r.t. $\mathcal{K}$, and let $\mathsf{cl}(q)$ be the set of all $C_{q'}$ such that $q' \in Q$. Note that this implies that $\mathsf{cl}(q)$ contains every concept name occurring in q.

An ABox $\mathcal{A}'$ is called a *q-completion* if it contains only assertions of the form

- $\neg C(a)$ for some $C \in \mathsf{cl}(q)$ and $a \in \mathsf{Ind}(\mathcal{A})$ and
- $\neg r(a, b)$ for a role name r occurring in $\mathsf{cl}(q)$ and $a, b \in \mathsf{Ind}(\mathcal{A})$.

Let $n = |q|, m_{\mathcal{H}} = |\mathcal{H}|, m_{\mathcal{A}} = |\mathcal{A}|$, and $k = |\mathsf{cl}(q)|$. By Lemmas 9 and 11 and since the number of sub-queries of q is bounded by 2^n, there is a polynomial p such that $k \leq 2^{p(n)\cdot \log p(m_{\mathcal{H}})}$. Also by Lemmas 9 and 11, there is a polynomial p' such that the size of each concept in $\mathsf{cl}(q)$ is bounded by $p'(n)$. Therefore, the number of q-completions is bounded by $2^{km_{\mathcal{A}}+2km_{\mathcal{A}}^2}$.

Let q' be a transitive rewriting of q, and $\tau : \mathsf{Var}(q') \to \mathsf{Ind}(\mathcal{A})$ be a partial mapping. For $a \in \mathsf{Ind}(\mathcal{A})$, we set $\mathsf{Root}(a, \tau) = \{v \in \mathsf{Var}(q') \mid \tau(v) = a\}$, and we use $\mathsf{Reach}(a, \tau)$ to denote the set of variables $v \in \mathsf{Var}(q')$ for which there exists a sequence of variables $v_0, \ldots, v_{n-1}$, $n \geq 1$, such that

- $\tau(v_0) = a$ and $v_{n-1} = v$
- $\{v_0, \ldots, v_{n-1}\} \cap \mathsf{dom}(\tau) \subseteq \mathsf{Root}(a, \tau)$;
- for all $i < n - 1$, there is a role r s.t. $r(v_i, v_{i+1}) \in q$.

Observe that $\mathsf{Root}(a, \tau) = \mathsf{dom}(\tau) \cap \mathsf{Reach}(a, \tau)$.

We call τ a *split mapping* if $\mathsf{dom}(\tau) \neq \emptyset$ and, for all $a, b \in \mathsf{Ind}(\mathcal{A})$, $a \neq b$ implies $\mathsf{Reach}(a, \tau) \cap \mathsf{Reach}(b, \tau) = \emptyset$. Each split mapping τ induces, for each $a \in \mathsf{ran}(\tau)$, a sub-query q_a as follows:

$$\begin{aligned} q_a = \ & \{at \in q \mid \mathsf{Var}(\{at\}) \subseteq \mathsf{Reach}(a, \tau)\} \setminus \\ & \{r(v, v') \in q' \mid v, v' \in \mathsf{Root}(a, \tau)\}. \end{aligned}$$

An *extended* query is a query where disjunctions of $\mathcal{ELI}^{\sqcap}$ concepts can occur in concept atoms. From a transitive rewriting $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ and a split mapping $\tau : \mathsf{Var}(q') \to \mathsf{Ind}(\mathcal{A})$ we obtain a *groundable rewriting* q'' of q' as follows:

- drop all atoms in q' which contain a variable $v \notin \mathsf{dom}(\tau)$;
- for each $a \in \mathsf{ran}(\tau)$, replace all variables $v \in \mathsf{Root}(a, \tau)$ with a new variable v_a; and
- for each $a \in \mathsf{ran}(\tau)$, let q_a be the sub-query of q' induced by τ, replace all $v \in \mathsf{Root}(a, \tau)$ with v_a and add $(C_{q_a^1} \sqcup \ldots \sqcup C_{q_a^m})(v_a)$, where each q_a^i is a tree transformation of q_a in which v_a was not replaced and $C_{q_a^i} = C_{q_a^i}(v_a)$.

In this case, we call τ the *grounding of* q'' and use $\tau(q'')$ for the result of replacing each v_a in q'' with a.

We say that a q-completion $\mathcal{A}'$ *spoils* $\tau(q'')$ if there is some

- $r(a, b) \in \tau(q'')$ and $\neg r(a, b) \in \mathcal{A}'$ or
- $(C_{q_a^1} \sqcup \ldots \sqcup C_{q_a^m})(a) \in \tau(q'')$ and $\neg C_{q_a^i}(a) \in \mathcal{A}'$ for $1 \leq i \leq m$.

Finally, a q-completion $\mathcal{A}'$ is called a *counter candidate* for q and $\mathcal{K}$ if, for all groundable rewritings q'' of transitive rewritings q' of q with grounding τ, $\mathcal{A}'$ spoils $\tau(q'')$.

Let us estimate the complexity of checking whether a given q-completion is a counter candidate. By Lemma 9, there is a polynomial p such that there are $2^{p(n) \cdot \log p(m_{\mathcal{H}})}$ transitive rewritings of q and it is easily seen that all tree transformations can be computed in this time bound as well. The number of q-completions (and therefore of counter candidates) is bounded by $2^{km_{\mathcal{A}} + 2km_{\mathcal{A}}^2}$. Moreover, for $q' \in \mathsf{tr}(q)$, it can be decided in time polynomial in n and $m_{\mathcal{A}}$ whether a partial mapping τ is a split mapping for q' and $\mathcal{A}$, and there are at most $(m_{\mathcal{A}} + 1)^{|q'|}$ such partial mappings. In order to check whether a q-completion is a counter candidate, we have to check for the existence of certain concepts C such that $C(a) \in \mathcal{A}'$. Clearly, the number of concepts relevant here is bounded by the cardinality of $\mathsf{cl}(q)$, which is bounded by $2^{p(n) \cdot \log p(m_{\mathcal{H}})}$ for a polynomial p. Together with Lemma 9, this implies that there is a polynomial p' such that checking whether a q-completion is a counter candidate can be done in time $2^{p(n) \cdot \log p(m_{\mathcal{H}}) \cdot \log p(m_{\mathcal{A}})}$.

The following lemma forms the base of our decision procedure.

Lemma 16. *$\mathcal{K} \not\models q$ iff there exists a counter candidate $\mathcal{A}'$ for q and $\mathcal{K}$ such that the extended knowledge base $\mathcal{K}' = (\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$ is consistent.*

Intuitively, counter candidates are those q-completions that do not give rise to true forest matches. Since we prevent tree matches via the TBox $\mathcal{T}_q$, the knowledge bases $\mathcal{K}'$ of Lemma 16 capture exactly the counter models against $\mathcal{K} \models q$.

Based on this lemma, we define two versions of our decision procedure for query entailment in $\mathcal{SHIQ}$. The first version is deterministic and provides us with an upper bound for combined complexity, where all three components of the input knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{H}, \mathcal{A})$ and the query are considered as the input. The second version is non-deterministic and yields a tight upper bound for data complexity, where $\mathcal{T}$, $\mathcal{H}$, and q are assumed fixed, and only $\mathcal{A}$ is the input. For the deterministic version, we make use of the following result.

Theorem 17. *Given an extended knowledge base $\mathcal{K}' = (\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$, where $|(\mathcal{T}, \mathcal{H}, \mathcal{A})| = r$, the cardinality of $\mathcal{T}_q \cup \mathcal{A}'$ is s, and the maximum length of concepts in $\mathcal{T}_q$ and $\mathcal{A}'$ is t, we can decide consistency of $\mathcal{K}$ in deterministic time $2^{2^{p(t \cdot \log r \cdot \log s)}}$ with p a polynomial.*

The deterministic version of our algorithm is as follows: generate all q-completions of $\mathcal{A}$ and then check whether all extended knowledge bases that are induced by the counter candidates are inconsistent. By Lemma 16, this algorithm is correct. Observe that for all extended knowledge bases $\mathcal{K}' = (\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$ whose inconsistency needs to be checked, the cardinality of $\mathcal{T}_q$ is bounded by k, the cardinality of $\mathcal{A}'$ is bounded by $km_{\mathcal{A}} + 2km_{\mathcal{A}}^2$, and hence the cardinality of $\mathcal{T}_q \cup \mathcal{A}'$ is bounded by $k + km_{\mathcal{A}} + 2km_{\mathcal{A}}^2$ (where $k = |\mathsf{cl}(q)|$), and (due to Parts (b) of Lemmas 9 and 11) the maximum length of concepts in $\mathcal{T}_q$ and $\mathcal{A}'$ is bounded by $p(n)$ for some polynomial p. This together with Theorem 17, the bound established above on the number of q-completions of $\mathcal{A}$, and the fact that deciding if a q-completion is a counter candidate can be checked in time $2^{p(n)\cdot\log p(m_{\mathcal{H}})\cdot\log p(m_{\mathcal{A}})}$ with p a polynomial, yields the following result.

Theorem 18. *Given a $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$ and a conjunctive query q with $|\mathcal{K}| = m$ and $|q| = n$, it can be decided in deterministic time $2^{2^{p(n)\cdot\log p(m)}}$ whether $\mathcal{K} \models q$, where p is a polynomial.*

Observe that this bound is single exponential in the size of the knowledge base and double exponential in the size of the query.

The non-deterministic version of our decision procedure actually decides *non*-entailment of queries: guess a q-completion of $\mathcal{A}$, check whether it is a counter candidate and consistent, return "yes" ($\mathcal{K} \not\models q$) if the check succeeds and "no" ($\mathcal{K} \models q$) otherwise. Regarding the complexity of inconsistency, we make use of the following result.

Theorem 19. *Let $\mathcal{T}$ and $\mathcal{T}_q$ be TBoxes and $\mathcal{H}$ a role hierarchy. Given ABoxes $\mathcal{A}$ and $\mathcal{A}'$ such that $\mathcal{K}' = (\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$ is an extended knowledge base and $|\mathcal{A} \cup \mathcal{A}'| = r$, we can decide in non-deterministic time $p(r)$ whether $\mathcal{K}'$ is consistent.*

Again, Lemma 16 yields correctness of our algorithm. Let $m_{\mathcal{A}} = |\mathcal{A}|$. The bound established above on the maximal size of q-completions implies that q-completions of $\mathcal{A}$ are polynomial in $m_{\mathcal{A}}$. Whether a q-completion is a counter candidate can be decided in time $2^{p(n)\cdot\log p(m_{\mathcal{H}})\cdot\log p(m_{\mathcal{A}})}$, which is also polynomial in $m_{\mathcal{A}}$. Thus, Theorem 19 implies that the data complexity of query entailment in $\mathcal{SHIQ}$ is in co-NP. The lower bound easily follows from the fact that conjunctive query entailment is already co-NP-hard regarding data complexity in the very restricted DL $\mathcal{AL}$ [18].

Theorem 20. *Conjunctive query entailment in $\mathcal{SHIQ}$ is data complete for co-*NP.

1.2.5 Challenges

The main challenge is to extend the algorithms presented in this document to more expressive logics such as $\mathcal{SHOIQ}$. The aim is also to implement these algorithms and provide suitable optimizations which make them practical.

1.3 Less Expressive DLs and Hardness Results

Querying DL knowledge bases has received great attention in the last years. Indeed, the definition of suitable query languages, and the design of query answering algorithms is arguably one of the crucial issues in applying DLs to ontology management and to the Semantic Web [28].

Answering queries in DLs must take into account the open-world semantics of such logics, and is therefore much more difficult than in databases. For example, while first-order logic (FOL) is the basis of any query language (e.g., relational algebra and SQL) for relational databases [2], it is well-known that answering FOL queries posed to DL knowledge bases is undecidable[3]. More precisely, to the best of our knowledge, the most expressive class of queries that go beyond instance checking, and for which decidability of query answering has been proved in DLs, is the class of union of conjunctive queries [19, 73, 74]. This restriction on the query language may constitute a serious limitation to the adoption of DLs technology in information management tasks, such as those required in Semantic Web applications.

In this section we consider queries specified over ontologies expressed in Description Logics, i.e., knowledge bases (KBs) constituted by a TBox and an ABox, and address the query answering problem under the open-world semantics of DLs, i.e., the problem of computing the answers to a query that are logical consequences of the TBox and the ABox. We consider query languages that are fragments of first-order logic, and in particular find out results for the languages of Conjunctive Queries (CQs) and Union of Conjunctive Queries (UCQs), i.e., the maximal fragments for which the problem is known to be decidable. We analyze the data complexity of the problem, i.e., computational complexity measured with respect to the size of the ABox only. Note that we borrow the notion of data complexity from the database literature [87], on the premise that an ABox can be naturally viewed as a relational database. Our interest in the data complexity of the problem is motivated by the growth of applications in which ontologies are used as a conceptual view over data repositories, and therefore the set of instances of the concepts in the ontology are very large, actually much larger than the intensional level of the ontology, and call for being managed in a secondary storage.

We are interested in characterizing the polynomial tractability boundaries of query answering, depending on the expressive power of the DL used to specify the KB. Furthermore, we want also to characterize the LogSpace boundary of the problem. This boundary is particularly important from a practical view point: staying in LogSpace actually allows us to reduce the query answering problem to evaluation of a first-order query (i.e., a query expressible in SQL) over a database which represents the ABox of the knowledge base (see Section 1.3.7 for more details on this aspect).

Our overall investigation on query answering is carried out within the entire Work Package (WP) 4 ("Ontology access, processing, and usage"), and is focalized on two main aspects: (*a*) establishing how difficult is the problem from a computational point of view, in order to get hints on the practical feasibility of query answering over DLs ontologies; (*b*) Defining techniques for query answering which allow for a simple and efficient implementation.

In the present report we concentrate on the first aspect, and show that query answering over DL knowledge bases is a hard task already for quite simple DLs. More precisely, the contributions of the present technical report are the following.

[3]Indeed, validity in FOL can be reduced to query answering.

- We show minimal DL languages for which the data complexity of query answering (in fact, of instance checking) is NLogSpace-hard and PTime-hard in all the above mentioned query languages. In spite of the fact that we conjecture that for such languages query answering is polynomially tractable (in NLogSpace and PTime, respectively), these hardness results tell us that in query answering we cannot take advantage of state-of-the-art database query optimization strategies, and this might hamper practical feasibility for very large ABoxes (see also Section 1.3.7).

- We then establish coNP-hardness of query answering with respect to data complexity for surprisingly simple DLs. In particular, we show that we get intractability as soon as the DL is able to express simple forms of union.

Once established the above complexity lower bounds, in the next months of the project, within WP 4, we will concentrate on the development of practical query answering techniques. We simply remark now that a particular interest will be posed on the definition and the study of those DLs for which answering queries specified in expressive query languages (namely, conjunctive queries) is in LogSpace, thus allowing for the exploitation of database technique, as mentioned above.

In the rest of this section we first introduce some formal definitions (Section 1.3.1), then we identify DLs for which query answering is NLogSpace-hard (Section 1.3.2), and DLs for which query answering is PTime-hard (Section 1.3.3). Finally we identify DLs for which query answering is coNP-hard (Section 1.3.4). In the last three subsections we overview related work, summarize presented results and discuss ongoing research on this topic.

The material included in the present document, which is our first outcome within the TONES project for WP 4, has been recently disseminated in international conferences [18].

1.3.1 Formal Definitions

Description Logics are logics that represent the domain of interest in terms of *concepts*, denoting sets of objects, and *roles*, denoting binary relations between (instances of) concepts. Complex concept and role expressions are constructed starting from a set of atomic concepts and roles by applying suitable constructs. Different DLs allow for different constructs. In this section, we distinguish between the constructs that are allowed in the concepts in the left-hand side (Cl) and in the concepts in the right-hand side (Cr) of inclusion assertions (see later).

A $\mathcal{L}_{DL}$ *knowledge base* (KB) $\mathcal{K} = (\mathbf{T}, \mathcal{A})$ represents the domain of interest and consists of two parts, a *TBox* $\mathbf{T}$, representing intensional knowledge, and an *ABox* $\mathcal{A}$, representing extensional knowledge, both specified according to the DL $\mathcal{L}_{DL}$. The TBox is formed by a set of *inclusion assertions* of the form

$$Cl \sqsubseteq Cr$$

where Cl and Cr are formed using the constructs allowed by $\mathcal{L}_{DL}$. Such an inclusion assertion expresses that all instances of concept Cl are also instances of concept Cr.

Possibly, the TBox allows for the specification of *functionality assertions* of the form

$$(\mathit{funct}\ R)$$

where R is either a role P or its inverse P^-. The ABox is formed by a set of *membership assertions* on atomic concepts and on atomic roles:

$$A(a), \qquad P(a_1, a_2)$$

stating respectively that the object (denoted by the constant) a is an instance of A and that the pair (a_1, a_2) of objects is an instance of the role P. We assume that in each DL $\mathcal{L}_{DL}$ considered in this section, membership assertions assume the above form, whereas they differ for constructs of the TBox.

We now specify the semantics of inclusion, functional and membership assertions. An interpretation $\mathcal{I} = (\mathsf{dom}, \cdot^{\mathcal{I}})$ is a *model* of an inclusion assertion $Cl \sqsubseteq Cr$ if $Cl^{\mathcal{I}} \subseteq Cr^{\mathcal{I}}$, where $Cl^{\mathcal{I}}$ and $Cl^{\mathcal{I}}$ is the standard concept interpretation in DLs [6], and depends on the particular DL $\mathcal{L}_{DL}$ adopted. Furthermore, $\mathcal{I}$ is a model of an assertion $(\mathit{funct}\ P)$ if the binary relation $P^{\mathcal{I}}$ corresponding to the interpretation of the role P is a function, i.e., $(o, o_1) \in P^{\mathcal{I}}$ and $(o, o_2) \in P^{\mathcal{I}}$ implies $o_1 = o_2$. Analogously for $(\mathit{funct}\ P^-)$.

To specify the semantics of membership assertions, we recall that the interpretation function assigns to each constant a a *distinct* object $a^{\mathcal{I}} \in \mathsf{dom}$. Note that this implies that, as usual in DLs, we enforce the *unique name assumption* on constants [6]. An interpretation $\mathcal{I}$ is a model of a membership assertion $A(a)$ (resp., $P(a_1, a_2)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}$). A *model of a KB* $\mathcal{K} = (\mathbf{T}, \mathcal{A})$ is an interpretation $\mathcal{I}$ that is a model of all assertions in $\mathbf{T}$ and $\mathcal{A}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* (an assertion) α, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

We can extract information from the extensional level of a KB $\mathcal{K}$ expressed in a DL $\mathcal{L}_{DL}$, by using queries expressed in a query language $\mathcal{L}_Q$. We consider in the following query languages which are fragments of FOL.

A query $q \in \mathcal{L}_Q$ over a $\mathcal{L}_{DL}$ KB $\mathcal{K}$ is therefore an open formula of first-order logic of the form

$$\{\, \vec{x} \mid \phi(\vec{x}) \,\}$$

where $\phi(\vec{x})$ is a FOL formula with free variables $\vec{x}$ and whose atoms are specified in terms of concepts and roles in $\mathcal{K}$. We call the size of $\vec{x}$ the arity of the query q. In the following, we simply denote a query q with free variables $\vec{x}$ with $q[\vec{x}]$. Given an interpretation $\mathcal{I}$, $q^{\mathcal{I}}$ is the set of tuples $\vec{o}$ of objects that, when assigned to the free variables, make the formula ϕ true in $\mathcal{I}$ [2].

The reasoning service we are interested in is *query answering*: given a $\mathcal{L}_{DL}$ knowledge base $\mathcal{K}$ and a $\mathcal{L}_Q$ query $q[\vec{x}]$ over $\mathcal{K}$, return all tuples $\vec{a}$ of constants in $\mathcal{K}$ such that, when substituted to the variables $\vec{x}$ in $q(\vec{x})$, we have that $\mathcal{K} \models q(\vec{a})$, i.e., such that $\vec{a}^{\mathcal{I}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.

We observe that query answering (properly) generalizes a well known reasoning service in DLs, namely *instance checking*, i.e., logical implication of an ABox assertion. In particular, instance checking can be expressed as the problem of answering (boolean) conjunctive queries constituted by just one ground atom.

Finally, we refer to *data complexity* of query answering, which is a notion borrowed from relational database theory [87]. First, we note that there is a recognition problem associated with query answering, which is defined as follows. We have a fixed TBox $\mathbf{T}$ expressed in a DL $\mathcal{L}_{DL}$, and a fixed query q expressed in a language $\mathcal{L}_Q$: the *recognition problem* associated to $\mathbf{T}$ and q is the decision problem of checking whether, given an ABox $\mathcal{A}$, and a tuple $\vec{a}$ of constants, we have that $(\mathbf{T}, \mathcal{A}) \models q(\vec{a})$. Note that neither the TBox nor the query is an input to the recognition problem.

Let $\mathcal{S}$ be a complexity class. When we say that query answering for a certain DL $\mathcal{L}_{DL}$ *is in* $\mathcal{S}$ *with respect to data complexity*, we mean that the corresponding recognition problem is in $\mathcal{S}$. Similarly, when we say that query answering for a certain DL $\mathcal{L}$ is $\mathcal{S}$*-hard with respect to data complexity*, we mean that the corresponding recognition problem is $\mathcal{S}$-hard.

The importance of studying query answering and the relevance of characterizing its data complexity is motivated by the recent growth of applications in which ontologies are used as a conceptual view over data repositories. For example, in Enterprise Application Integration [57], Data Integration [58], and the Semantic Web [47], the intensional level of the application domain can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system. In these contexts, the set of instances of the concepts in the ontology is to be managed in the data layer of the system architecture (e.g., in the lowest of the three tiers of the Enterprise Software Architecture), and, since instances correspond to the data items of the underlying information system, such a layer constitutes a very large (much larger than the intensional level of the ontology) repository, to be stored in secondary storage (see [14]).

When clients access the application ontology, it is very likely that one of the main services they need is the one of answering complex queries over the extensional level of the ontology (obviously making use of the intensional level as well in producing the answer). Given the size of the instance repository, when measuring the computational complexity of query answering (and reasoning in general) the most important parameter is the size of the data, i.e., its *data complexity*.

1.3.2 NLogSpace-hard DLs

We now consider basic DLs for which query answering is NLogSpace-hard. In fact we show that already the problem of instance checking for such DLs is NLogSpace-hard. As a consequence we get NLogSpace-hardness of query answering when $\mathcal{L}_Q$ is the language of conjunctive queries or union of conjunctive queries.

The first case we consider is that of a basic DL language, which allows for qualified existential quantification in the right-hand side of inclusion assertions. The second case is that of a basic DL language which allows for qualified universal quantification in the right-hand side of inclusion assertions. The third case that we consider is that of a language which allows for qualified existential quantification in the right-hand side of inclusion assertions, together with the possibility of expressing functionality constraints. This is formally stated in the following theorem.

Theorem 21. *Instance checking is* NLogSpace*-hard with respect to data complexity for the cases where*

1. $$\begin{array}{lcl} Cl & \rightarrow & A \mid \exists R.A \\ Cr & \rightarrow & A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr$

2. $$\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid \forall R.A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr$

3. $$\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid \exists R.A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr, (\mathit{funct}\ R)$

Proof. For Case 1, the proof is by a LogSpace reduction from reachability in directed graphs, which is NLogSpace-complete. Let $G = (N, E)$ be a directed graph, where N is a set of nodes and $E \subseteq N \times N$ is the set of edges of G, and let s, d be two nodes in N. Reachability is the problem of checking whether there is a path in G from s to d.

We define a KB $\mathcal{K} = (\mathbf{T}, \mathcal{A})$, where the TBox $\mathbf{T}$ is constituted by a single inclusion assertion

$$\exists P.A \sqsubseteq A$$

and the ABox $\mathcal{A}$ has as constants the nodes of G, and is constituted by the membership assertion $A(d)$, and by one membership assertion $P(n, n')$ for each edge $(n, n') \in E$. It is easy to see that $\mathcal{K}$ can be constructed in LogSpace from G, s, and d. We show that there is a path in G from s to d if and only if $\mathcal{K} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in G from s to d. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\mathsf{dom} = N$, $n^{\mathcal{I}} = n$ for each $n \in N$, $P^{\mathcal{I}} = E$, and $A^{\mathcal{I}} = \{ n \mid \text{ there is a path in } G \text{ from } n \text{ to } d \}$. We show that $\mathcal{I}$ is a model of $\mathcal{K}$. By construction, $\mathcal{I}$ satisfies all membership assertions $P(n, n')$ and the membership assertion $A(d)$. Consider an object $n \in (\exists P.A)^{\mathcal{I}}$. Then there is an object $n' \in A^{\mathcal{I}}$ such that $(n, n') \in P^{\mathcal{I}}$. Then, by definition of $\mathcal{I}$, there is a path in G from n' to d, and $(n, n') \in E$. Hence, there is also a path in G from n to d and, by definition of $\mathcal{I}$, we have that $n \in A^{\mathcal{I}}$. It follows that also the inclusion assertion $\exists P.A \sqsubseteq A$ is satisfied in $\mathcal{I}$.

"$\Rightarrow$" Suppose there is a path in G from a node n to d. We prove by induction on the length k of such a path that $\mathcal{K} \models A(n)$. Base case: $k = 0$, then $n = d$, and the claim follows from $A(d) \in \mathcal{A}$. Inductive case: suppose there is a path in G of length $k - 1$ from n' to d and $(n, n') \in E$. By the inductive hypothesis, $\mathcal{K} \models A(n')$, and since by definition $P(n, n') \in \mathcal{A}$, we have that $\mathcal{K} \models \exists P.A(n)$. By the inclusion assertion in $\mathbf{T}$ it follows that $\mathcal{K} \models A(n)$.

For Case 2, the proof follows from Case 1 and the observation that an assertion $\exists P.A_1 \sqsubseteq A_2$ is logically equivalent to the assertion $A_1 \sqsubseteq \forall P^-.A_2$, and that we can get rid of inverse roles by inverting the edges of the graph represented in the ABox.

For Case 3, the proof is again by a LogSpace reduction from reachability in directed graphs, and is based on the idea that an assertion $\exists P.A_1 \sqsubseteq A_2$ can be simulated by the assertions $A_1 \sqsubseteq \exists P^{-}.A_2$ and (*funct* P^{-}). Moreover, the graph can be encoded using only functional roles, and we can again get rid of inverse roles by inverting edges.

More precisely, let $G = (N, E)$ be a directed graph and consider the problem of reachability in G between nodes s and d. We define the KB $\mathcal{K} = (\mathbf{T}, \mathcal{A})$, where the TBox $\mathbf{T}$ is constituted by the inclusion assertions

$$A \sqsubseteq \exists P_1.B \qquad B \sqsubseteq \exists P_1.B \qquad B \sqsubseteq \exists P_2.A \qquad (\textit{funct } P_1) \qquad (\textit{funct } P_2)$$

and the ABox $\mathcal{A}$ makes use of the nodes in N and the edges in E as constants. Consider a node n of G, and let $e_1, \ldots, e_k$ be all edges of G that have n as their target (i.e., such that $e_i = (n_i, n)$ for some node n_i), taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_1(n, e_1)$, and $P_1(e_i, e_{i+1})$ for $i \in \{1, \ldots, k-1\}$,
- $P_2(e_i, n_i)$, where $e_i = (n_i, n)$, for $i \in \{1, \ldots, k-1\}$.

Additionally, $\mathcal{A}$ contains the membership assertion $A(d)$. Notice that the assertions in the ABox do not violate the functionality assertions in the TBox. Again, it is easy to see that $\mathcal{K}$ can be constructed in LogSpace from G, s, and d. We show that there is a path in G from s to d if and only if $\mathcal{K} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in G from s to d. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\mathsf{dom} = \{o\} \cup N \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^{\mathcal{I}}$ and $P_2^{\mathcal{I}}$ contain all pairs of nodes directly required by the ABox assertions, $A^{\mathcal{I}}$ contains each node n such that there is a path in G from n to d, and $B^{\mathcal{I}}$ contains all edges (i, j) such that there is a path in G from j to d. To satisfy the assertion $A \sqsubseteq \exists P_1.B$ for those objects $n \in A^{\mathcal{I}}$ that have no outgoing P_1 edge forced by the ABox (i.e., that have no incoming edge in G), we set $o \in B^{\mathcal{I}}$, $(n, o) \in P_1^{\mathcal{I}}$, and $(o, o) \in P_1^{\mathcal{I}}$. We use o in a similar way to satisfy the assertions $B \sqsubseteq \exists P_1.B$ and $B \sqsubseteq \exists P_1.A$. Note that in this way the functionality assertions are not violated. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{K}$, and since there is no path in G from s to d, we have that $s \notin A^{\mathcal{I}}$.

"$\Rightarrow$" Suppose there is a path in G from a node n to d. We prove by induction on the length ℓ of such a path that $\mathcal{K} \models A(n)$. Base case: $\ell = 0$, then $n = d$, and the claim follows from $A(d) \in \mathcal{A}$. Inductive case: suppose there is a path in G of length $\ell - 1$ from j to d and $(n, j) \in E$. Let $n_1, \ldots, n_h$ be the nodes of G such that $(n_i, j) \in E$, up to $n_h = n$ and in the same order used in the construction of the ABox. By the inductive hypothesis, $\mathcal{K} \models A(j)$, and by the assertion $A \sqsubseteq \exists P_1.B$, functionality of P_1, and the ABox assertion $P_1(j, (n_1, j))$, we obtain that $\mathcal{K} \models B((n_1, j))$. Exploiting $B \sqsubseteq \exists P_1.B$, functionality of P_1, and the ABox assertion $P_1((n_i, j), (n_{i+1}, j))$, we obtain by induction on h that $\mathcal{K} \models B((n_h, j))$. Finally, by $B \sqsubseteq \exists P_2.A$, functionality of P_2, and the ABox assertion $P_2((n_h, j), n_h)$, we obtain that $\mathcal{K} \models A(n_h)$, i.e., $\mathcal{K} \models A(n)$. ❑

We remark again that all the above "negative" results hold for instance checking already, i.e., for the simplest queries possible. Also, note that in all three cases, we are considering minimal DLs (in terms of number of constructs allowed) for which NLOGSPACE-hardness of query answering is reached.

1.3.3 PTIME-hard DLs

Next we show that if we consider further extensions to the logics mentioned in Theorem 21, we get even stronger complexity results. In particular, we consider different cases where query answering (actually, instance checking already) becomes PTIME-hard in data complexity.

Note that the PTIME-hardness result basically means that we need at least the power of full Datalog to answer queries in these cases.

Theorem 22. *Instance checking (and hence query answering) is* PTIME-*hard with respect to data complexity for the cases where*

1. $$\begin{array}{lcl} Cl & \rightarrow & A \mid \exists R.A \\ Cr & \rightarrow & A \mid \exists P \\ R & \rightarrow & P \mid P^- \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr$

2. $$\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid \exists R.A \\ R & \rightarrow & P \mid P^- \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr, (\mathit{funct}\ R)$

3. $$\begin{array}{lcl} Cl & \rightarrow & A \mid \exists R.A \\ Cr & \rightarrow & A \mid \exists R.A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr, (\mathit{funct}\ R)$

Proof. For Case 1, we reduce the emptiness problem of context-free grammars to query answering over such DL-KBs. Let $G = \langle \mathsf{V}_N, \mathsf{V}_T, S, \mathcal{P} \rangle$ be a context-free grammar (the non-terminal symbol S is the axiom of G). Without loss of generality, we can assume that each production rule has at most two nonterminal symbols in its right-hand side, since each rule with more than two nonterminal symbols in its right-hand side can be linearly transformed into an equivalent set of production rules with at most two nonterminal symbols in their right-hand side. Let $\mathcal{L}(G)$ be the language generated by G.

Given a production rule R, we denote by $\mathit{Left}(R)$ the nonterminal symbol occurring in the left-hand side of R, and denote by $\mathit{Right}(R)$ the set of nonterminal symbols occurring in the right-hand side of R.

We define the KB $\mathcal{K} = \langle \mathbf{T}, \mathcal{A} \rangle$, where:

- The TBox $\mathbf{T}$ is constituted by the following inclusion assertions:

$$\begin{array}{rl} \exists L.D \sqsubseteq D & (I1) \\ \exists R.D \sqsubseteq A_1 & (I2) \\ \exists R.A_1 \sqsubseteq A_2 & (I3) \\ \exists P.A \sqsubseteq D & (I4) \\ \exists P^-.A_2 \sqsubseteq A & (I5) \\ A_1 \sqsubseteq \exists P & (I6) \end{array}$$

- The ABox $\mathcal{A}$ is constructed in the following way.

```
begin
  $\mathcal{A} := \emptyset$;
  $j = 1$;
  for each $N \in V_N$ do
  begin
    i=1;
    for each production rule $PR$ in $\mathcal{P}$ such that $Left(PR) = N$ do
    begin
      $\mathcal{A} := \mathcal{A} \cup \{L(n_i, n_{i+1})\}$;
      if $Right(PR) = \emptyset$
      then $\mathcal{A} := \mathcal{A} \cup \{D(N_i)\}$
      else if $Right(PR) = \{B\}$
      then begin $\mathcal{A} := \mathcal{A} \cup \{A_1(j), R(N_i, j), L(j, B_1)\}$; $j := j + 1$ end
      else if $Right(PR) = \{B, C\}$
      then begin
        $\mathcal{A} := \mathcal{A} \cup \{R(N_i, j), L(j, B_1), R(j, j + 1), L(j + 1, C_1)\}$;
        $j := j + 2$
      end;
      $i := i + 1$
    end
  end
end
```

It is immediate to see that $\mathcal{A}$ is constructed in time linear in the size of $\mathcal{P}$. (Notice that for each nonterminal symbol A, the individuals $a_1, \ldots, a_k$ represent the k occurrences of A in the left-hand sides of production rules in $\mathcal{P}$, while there is a distinct (new) individual j for each right-hand side occurrence of A in $\mathcal{P}$).

Finally, let q be the query $q :\!- D(S_1)$. We prove that $\mathcal{L}(G)$ is empty iff $\mathcal{K} \models q$. More precisely, we prove that, for every nonterminal symbol $A \in \mathsf{V}_N$, $\mathcal{K} \models D(A)$ iff A generates a non-empty language in G.

($\Leftarrow$) Suppose A generates a non-empty language in G. We prove that $\mathcal{K} \models D(a_1)$. The proof is by induction on the structure of a derivation of s from A in G. Base case (direct derivation): there exists a production rule such that $Left(PR) = A$ and $Right(PR) = \emptyset$. By the above definition of $\mathcal{A}$, $D(a_1) \in \mathcal{A}$, consequently $\mathcal{K} \models D(a_1)$.

Inductive case (indirect derivation): there exists a production rule such that $\mathit{Left}(PR) = A$ and each nonterminal symbol occurring in $\mathit{Right}(PR)$ generates a non-empty language in G. Suppose $\mathit{Right}(PR) = \{B, C\}$ (the case when $\mathit{Right}(PR) = \{B\}$ is analogous). By the inductive hypothesis, it follows that $\mathcal{K} \models D(b_1)$ and $\mathcal{K} \models D(c_1)$. Moreover, by definition of $\mathcal{A}$, there exist individuals i and j such that $R(a_k, i) \in \mathcal{A}$ for some k, $R(i, j) \in \mathcal{A}$, $L(i, b_1) \in \mathcal{A}$, $L(j, c_1) \in \mathcal{A}$. Since $\mathcal{K} \models D(c_1)$, from inclusion (I1) of **T** it follows that $\mathcal{K} \models D(j)$, consequently from inclusion (I2) it follows that $\mathcal{K} \models A_1(i)$, and from inclusion (I3) we have that $\mathcal{K} \models A_2(a_k)$; moreover, from $\mathcal{K} \models D(b_1)$ and inclusion (I1) it follows that $\mathcal{K} \models D(i)$, thus, from (I2) it follows that $\mathcal{K} \models A_1(a_k)$. Now, from inclusion (I6) we have that there exists individual ℓ such that $\mathcal{K} \models P(a_k, \ell)$, thus, from $\mathcal{K} \models A_2(a_k)$ and from inclusion (I5) it follows that $\mathcal{K} \models A(\ell)$, therefore $\mathcal{K} \models \exists P.A(a_k)$, and by inclusion (I4) it follows that $\mathcal{K} \models D(a_k)$.

($\Rightarrow$) Suppose that A generates a empty language in G. We prove that $\mathcal{K} \not\models D(a_1)$. The proof is by induction on the structure of G. The key property is that, from the definition of $\mathcal{A}$ it follows that, for each (new) individual i in $\mathcal{A}$ corresponding to a right-hand side occurrence of the nonterminal symbol A, $\mathcal{K} \models D(i)$ if and only if $\mathcal{K} \models D(a_1)$, since the concept D "propagates backward" only through the role L, and by definition of $\mathcal{A}$, each new individual i (representing a right-hand side occurrence of A) is connected through role L only to the individual A.

For Case 2, the reduction (and the proof of its correctness) is the same as in Case 1, with the exception of the TBox assertions which are the following:

$$\begin{array}{rl}
D \sqsubseteq \exists L^-.D & (I1) \\
D \sqsubseteq \exists R^-.A_1 & (I2) \\
A_1 \sqsubseteq \exists R^-.A_2 & (I3) \\
A \sqsubseteq \exists P^-.D & (I4) \\
A_2 \sqsubseteq \exists P.A & (I5) \\
A_1 \sqsubseteq \exists P & (I6) \\
(\mathit{funct}\ L^-) & (I7) \\
(\mathit{funct}\ R^-) & (I8) \\
(\mathit{funct}\ P) & (I9)
\end{array}$$

Also for Case 3, the reduction (and the proof of its correctness) is the same as in Case 1, with the exception of the TBox assertions which are the following:

$$\begin{array}{rl}
\exists L.D \sqsubseteq D & (I1) \\
\exists R.D \sqsubseteq A_1 & (I2) \\
\exists R.A_1 \sqsubseteq A_2 & (I3) \\
\exists P.A \sqsubseteq D & (I4) \\
A_2 \sqsubseteq \exists P.A & (I5) \\
A_1 \sqsubseteq \exists P & (I6) \\
(\mathit{funct}\ P) & (I7)
\end{array}$$

❏

Theorem 23. *Instance checking (and hence query answering) is* PTime*-hard with respect to data complexity for the cases where*

1. $$\begin{array}{lcl} Cl & \rightarrow & A \mid \exists R.A \mid A_1 \sqcap A_2 \\ Cr & \rightarrow & A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr$

2. $$\begin{array}{lcl} Cl & \rightarrow & A \mid A_1 \sqcap A_2 \\ Cr & \rightarrow & A \mid \forall R.A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr$

3. $$\begin{array}{lcl} Cl & \rightarrow & A \mid A_1 \sqcap A_2 \\ Cr & \rightarrow & A \mid \exists R.A \\ R & \rightarrow & P \end{array}$$
   TBox assertions: $Cl \sqsubseteq Cr, (\mathit{funct}\ R)$

Proof. For Case 1, the proof is by a LogSpace reduction from Path System Accessibility, which is PTime-complete [32]. An instance of Path System Accessibility is defined as $PS = (N, E, S, t)$, where N is a set of nodes, $E \subseteq N \times N \times N$ is an accessibility relation (we call its elements edges), $S \subseteq N$ is a set of source nodes, and $t \in N$ is a terminal node. PS consists in verifying whether t is *accessible*, where a node $n \in N$ is accessible if $n \in S$ or if there exist accessible nodes n_1 and n_2 such that $(n, n_1, n_2) \in E$.

We define the KB $\mathcal{K} = (\mathbf{T}, \mathcal{A})$, where the TBox $\mathbf{T}$ is constituted by the inclusion assertions

$$\exists P_1.A \sqsubseteq B_1 \qquad \exists P_2.A \sqsubseteq B_2 \qquad B_1 \sqcap B_2 \sqsubseteq A \qquad \exists P_3.A \sqsubseteq A$$

and the ABox $\mathcal{A}$ makes use of the nodes in N and the edges in E as constants. Consider a node $n \in N$, and let $e_1, \ldots, e_k$ be all edges in E that have n as their first component, taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_3(n, e_1)$, and $P_3(e_i, e_{i+1})$ for $i \in \{1, \ldots, k-1\}$,
- $P_1(e_i, j)$ and $P_2(e_i, k)$, where $e_i = (n, j, k)$, for $i \in \{1, \ldots, k-1\}$.

Additionally, $\mathcal{A}$ contains one membership assertion $A(n)$ for each node $n \in S$. Again, it is easy to see that $\mathcal{K}$ can be constructed in LogSpace from PS. We show that t is accessible in PS if and only if $\mathcal{K} \models A(t)$.

"$\Leftarrow$" Suppose that t is not accessible in PS. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $t^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\mathsf{dom} = N \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^{\mathcal{I}}$, $P_2^{\mathcal{I}}$, and $P_3^{\mathcal{I}}$ consist of all pairs of nodes directly required by the ABox assertions, $B_1^{\mathcal{I}}$ consists of all edges (i, j, k) such that j is accessible in PS, $B_2^{\mathcal{I}}$ consists of all edges (i, j, k) such that k is accessible in PS, and $A^{\mathcal{I}}$ consists of all nodes n that are accessible in PS union all edges (i, j, k) such that both j and k are accessible in PS. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{K}$, and since t is not accessible in PS, we have that $t \notin A^{\mathcal{I}}$.

"$\Rightarrow$" Suppose that t is accessible in PS. We prove by induction on the structure of the derivation of accessibility that if a node n is accessible, then $\mathcal{K} \models A(n)$. Base case (direct derivation): $n \in S$, hence, by definition, $\mathcal{A}$ contains the assertion $A(n)$ and $\mathcal{K} \models A(n)$. Inductive case (indirect derivation): there exists an edge $(n, j, k) \in E$ and both j and k are accessible. By the inductive hypothesis, we have that $\mathcal{K} \models A(j)$ and $\mathcal{K} \models A(k)$. Let $e_1, \ldots, e_h$ be the edges in E that have n as their first component, up to $e_h = (n, j, k)$ and in the same order used in the construction of the ABox. Then, by $P_1(e_h, j)$ in the ABox and the assertions $\exists P_1.A \sqsubseteq B_1$ we have that $\mathcal{K} \models B_1(e_h)$. Similarly, we get $\mathcal{K} \models B_2(e_h)$, and hence $\mathcal{K} \models A(e_h)$. By exploiting assertions $P_3(e_i, e_{i+i})$ in the ABox, and the TBox assertion $\exists P_3.A \sqsubseteq A$, we obtain by induction on h that $\mathcal{K} \models A(e_1)$. Finally, by $P_3(n, e_1)$, we obtain that $\mathcal{K} \models A(n)$.

For Cases 2 and 3, the proof follows from Case 1 and observations analogous to the ones for Theorem 21. ❑

1.3.4 coNP-hard DLs

Finally, we show three cases where the TBox language becomes so expressive that the data complexity of query answering goes beyond PTIME (assuming PTIME $\neq$ NP).

Theorem 24. *Query answering is coNP-hard with respect to data complexity for the cases where*

1. $Cl \rightarrow A \mid \neg A$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$

2. $Cl \rightarrow A$
   $Cr \rightarrow A \mid A_1 \sqcup A_2$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$

3. $Cl \rightarrow A \mid \forall R.A$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$

Proof. In all three cases, the proof is an adaptation of the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ presented in [25]. In the following, we first consider Case 2.

coNP-hardness of query answering is proved by a reduction from $2+2$-CNF unsatisfiability (which is showed to be coNP-complete in [25]). A $2+2$-CNF formula on an alphabet P is a CNF formula in which each clause has exactly four literals: two positive and two negative ones, where the propositional letters are elements of $P \cup \{\mathit{true}, \mathit{false}\}$. Given a $2+2$-CNF formula $F = C_1 \wedge \ldots \wedge C_n$, where $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$, we associate with it a knowledge base $\mathcal{K}_F = (\mathbf{T}_F, \mathcal{A}_F)$ and a query Q as follows. $\mathcal{K}_F$ has one constant ℓ for each letter L in F, one constant c_i for each clause C_i, plus two constants *true* and *false* for the corresponding propositional constants. The atomic roles of $\mathcal{K}_F$ are P_1, P_2, N_1, N_2 and the atomic concepts are O, A_t, and A_f. Then, we pose

$$\begin{array}{lcl}
\mathbf{T}_F & = & \{O \ \sqsubseteq A_t \cup A_f\}, \\
\mathcal{A}_F & = & \{\ A_t(\mathit{true}), A_f(\mathit{false}) \\
 & & O(\ell^1_{1+}), O(\ell^1_{2+}), O(\ell^1_{1-}), O(\ell^1_{2-}), \\
 & & \ldots \\
 & & O(\ell^n_{1+}), O(\ell^n_{2+}), O(\ell^n_{1-}), O(\ell^n_{2-}), \\
 & & P_1(c_1, \ell^1_{1+}), P_2(c_1, \ell^1_{2+}), N_1(c_1, \ell^1_{1-}), N_2(c_1, \ell^1_{2-}), \\
 & & \ldots \\
 & & P_1(c_n, \ell^n_{1+}), P_2(c_n, \ell^n_{2+}), N_1(c_n, \ell^n_{1-}), N_2(c_n, \ell^n_{2-})\ \},\ \text{and} \\
Q & = & \{\ \mid \exists\, x, y, z, w_1, w_2. P_1(x,y)A_f(y)P_2(x,z)A_f(z)N_1(x,w_1)A_t(w_1)N_2(x,w_2)A_t(w_2)\}.
\end{array}$$

Intuitively, the membership to the extension of A_f or A_t corresponds to the truth values *true* and *false* respectively and checking $\mathcal{K}_F \models Q$ (i.e., the query evaluates to true in $\mathcal{K}_F$) corresponds to checking whether in every truth assignment for the formula F there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e., a clause that is not satisfied. Note that the ABox $\mathcal{A}_F$ contains the assertions $A_t(\mathit{true})$ and $A_f(\mathit{false})$ in order to guarantee that in each model $\mathcal{I}$ of $\mathcal{K}_F$ the constants *true* and *false* are in the extension of (possibly both) $A_t^{\mathcal{I}}$ and $A_f^{\mathcal{I}}$, respectively.
Now, it remains to prove that the formula F is unsatisfiable if and only if $\mathcal{K}_F \models Q$.

"$\Leftarrow$" Suppose that F is unsatisfiable. Consider a model $\mathcal{I}$ of $\mathcal{K}_F$ (which always exists since $\mathcal{K}_F$ is always satisfiable), and let $\delta_{\mathcal{I}}$ be the truth assignment for F such that $\delta_{\mathcal{I}}(\ell) = \mathit{true}$ iff $\ell^{\mathcal{I}} \in A^{\mathcal{I}}$, for every letter ℓ in F (and corresponding constant in $\mathcal{K}_F$). Since F is unsatisfiable, there exists a clause C_i that is not satisfied by $\delta_{\mathcal{I}}$, and therefore $\delta_{\mathcal{I}}(L^i_{1+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{2+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{1-}) = \mathit{true}$ and $\delta_{\mathcal{I}}(L^i_{2-}) = \mathit{true}$. It follows that in $\mathcal{K}_F$ the interpretation of the constants related to c_i through the roles P_1 and P_2 is not in $A_t^{\mathcal{I}}$, and consequently is in the $A_f^{\mathcal{I}}$, and the interpretation of constants related to c_i through the roles N_1 and N_2 is in $A_t^{\mathcal{I}}$. Thus, there exists a substitution σ which assigns variables in Q to constants in $\mathcal{K}_F$ in such a way that $\sigma(Q)$ evaluates to true in $\mathcal{I}$ (notice that this holds even if the propositional constants *true* or *false* occur in F). Therefore, since this argument holds for each model $\mathcal{I}$ of $\mathcal{K}_F$, we can conclude that $\mathcal{K}_F \models Q$.

"$\Rightarrow$" Suppose that F is satisfiable, and let δ be a truth assignment satisfying F. Let $\mathcal{I}_\delta$ be the interpretation for $\mathcal{K}_F$ defined as follows:

- $O^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid \ell \text{ occurs in } F\}$,
- $A_t^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid \delta(\ell) = \mathit{true}\} \cup \{\mathit{true}\}$,
- $A_f^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid \delta(\ell) = \mathit{false}\} \cup \{\mathit{false}\}$,
- $\rho^{\mathcal{I}_\delta} = \{(a^{\mathcal{I}_\delta}, b^{\mathcal{I}_\delta}) \mid \rho(a,b) \in \mathcal{A}_F\}$ for $\rho = P_1, P_2, N_1, N_2$.

It is easy to see that $\mathcal{I}_\delta$ is a model of $\mathcal{K}_F$. On the other hand, since F is satisfiable, for every clause in F there exists a positive literal interpreted as true or a negative literal interpreted as false. It follows that for every constant c_i, there exists either a role (P_1 or P_2) that relates c_i to a constant whose interpretation is in $A_t^{\mathcal{I}_\delta}$ or there exists a role (N_1 or N_2) that relates c_i to a constant whose interpretation is in $A_f^{\mathcal{I}_\delta}$. Since the query Q is evaluated to true in $\mathcal{I}_\delta$ only if there exists at least a constant c_i in $\mathcal{K}_F$ such that the interpretations of the constants related to c_i by roles P_1 and P_2 are both in $A_f^{\mathcal{I}_\delta}$ and the interpretations of the constants related to c_i by roles N_1 and N_2 are both in $A_t^{\mathcal{I}_\delta}$, it follows that the query Q evaluates to false in $\mathcal{I}_\delta$ and therefore $\mathcal{K}_F \not\models Q$.

Proofs for Cases 1 and 3 are obtained by analogous reductions from $2+2$-CNF unsatisfiability. More precisely, for Case 1 the knowledge base $\mathcal{K}_F = (\mathbf{T}_F, \mathcal{A}_F)$ has the same constants and the same atomic roles as for Case 2, and has only the atomic concepts A_t and A_f. Then, $\mathbf{T}_F = \{\neg A_t \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 2 but without the assertions involving the concept O. Finally, the query Q is as for Case 2. For Case 3, $\mathcal{K}_F$ has the same constants as for Cases 1 and 2, the same atomic roles as for Cases 1 and 2 plus the atomic role P, and the atomic concepts A and A_f. Then, $\mathbf{T}_F = \{\forall P.A \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 1 but without the assertion $A_t(\mathit{true})$, which is substituted by the assertion $P(\mathit{true}, d)$, where d is a new constant not occurring elsewhere in $\mathcal{K}_F$. Finally, the query Q is as follows

$$\begin{array}{rcl} Q & = & \{ \mid \exists\, x, y, z, w_1, w_2.P_1(x, y)A_f(y)P_2(x, z)A_f(z)N_1(x, w_1) \\ & & P(w_1, w_2)N_2(x, w_2)P(w_3, w_4)\}. \end{array}$$

Soundness and completeness of the above reductions can be proved as done for the reduction of Case 2. We finally point out that the intuition behind the above results is that in all three cases it is possible to require a reasoning by case analysis, caused by set covering assertions. Indeed, whereas in Case 2 we have explicitly asserted $O \sqsubseteq A_t \sqcup A_f$, for the other cases this can be seen by considering that A_t and A_f, and $\forall P.A$ and $\exists P$ cover the entire domain in Case 1 and Case 3, respectively. ❏

1.3.5 Related Work

All the DLs studied in this section are fragments of expressive DLs with assertions and inverses studied in the 90's (see [6] for an overview), which are at the base of current ontology languages such as OWL, and for which optimized automated reasoning systems such as FaCT++ [86], RacerPro [39] and Pellet [83] have been developed. Indeed, one could use, off-the-shelf, a system like RacerPro or Pellet to perform instance checking in such DLs. Also, reasoning with conjunctive queries in these DLs has been studied (see e.g., [19, 20]), although not yet implemented in systems. Unfortunately, the known reasoning algorithms for these DLs are in 2EXPTIME with respect to combined complexity, and more importantly they are not tailored towards obtaining tight complexity bounds with respect to data complexity (they are in EXPTIME). Alternative reasoning procedures that allow for clearly isolating data complexity have recently been proposed, how they will work in practice still needs to be understood. A coNP upper bound for data complexity of instance checking in the expressive DL $\mathcal{SHIQ}$ has been shown by making use of a reduction to Disjunctive Datalog and then exploiting resolution [54, 55]. It remains open whether such a technique can be extended to deal efficiently with conjunctive queries. In [60], making use of an algorithm based on tableaux, a coNP, upper-bound with respect to data complexity is given for a DL with arbitrary inclusion assertions, but lacking inverse roles. Recently, building on such techniques, coNP-completeness of answering conjunctive queries for $\mathcal{SHIQ}$, which includes inverse roles, and number restrictions (that generalize functionality) has been shown [75]. It is interesting to observe that the results in this section (Theorem 24) tell us that we get coNP-completeness already for very small fragments of $\mathcal{SHIQ}$.

In [55], a fragment of $\mathcal{SHIQ}$, called Horn-$\mathcal{SHIQ}$, is studied and a PTime upper bound in data complexity for instance checking is shown. The results in this section (Theorem 22) tell us that instance checking in Horn-$\mathcal{SHIQ}$ is also PTime-hard. Indeed, Horn-$\mathcal{SHIQ}$ allows for qualified existential quantification $\exists P.A$ in both sides of inclusion assertions and (an extended form) of functionality restrictions.

Finally, since DLP [38] is a superset of the DL in Case 1 of Theorem 23, then such a theorem shows that query answering in DLP is PTime-hard.

1.3.6 Summary of Results

Results provided in this section are summarized in the following table.

Cl	*Cr*	CQs (and therefore UCQs)	FOL
$A \mid \exists P.A$	A	NLogSpace-hard*	undecidable
A	$A \mid \forall P.A$	NLogSpace-hard*	undecidable
A	$A \mid \exists P.A$	NLogSpace-hard*	undecidable
$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	PTime-hard*	undecidable
A	$A \mid \exists P.A \mid \exists P^-.A$	PTime-hard*	undecidable
$A \mid \exists P.A$	$A \mid \exists P.A$	PTime-hard*	undecidable
$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	PTime-hard*	undecidable
$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	PTime-hard*	undecidable
$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	PTime-hard*	undecidable
$A \mid \neg A$	A	coNP-hard	undecidable
A	$A \mid A_1 \sqcup A_2$	coNP-hard	undecidable
$A \mid \forall P.A$	A	coNP-hard	undecidable

* This result holds already for instance checking.

Legenda: A (possibly with subscript) = atomic concept, P = atomic role, Cl/Cr = left/right-hand side of inclusion assertions, CQs = conjunctive queries, UCQs = union of conjunctive queries, FOL = first-order queries

In the table we report data complexity lower bounds for query answering over knowledge bases specified in the DLs studied in this section, when queries belong to expressive query languages such as the language of conjunctive queries, and the language of union of conjunctive queries. Undecidability of query answering for FOL follows from the fact that validity of a FOL formula ϕ can trivially be reduced to query answering (ϕ is a boolean query over an empty KB).

1.3.7 Discussion

In previous sections we have presented first fundamental results on the data complexity (complexity with respect to the size of the ABox only) of query answering in DLs. In particular, we have concentrated on the lower bounds of the problem, and have shown that answering expressive queries over (simple) DLs is a hard task from the computational complexity view point, since we easily reach intractability.

We have also investigated the LogSpace boundary of the problem, and we have singled out those DLs for which query answering becomes NLogSpace-hard and PTime-hard respectively. This boundary is particularly interesting from a practical view point: staying in LogSpace actually allows us to look for a way of encoding the problem by making use of first-order logic. More precisely, such a computational characterization allows us to reduce query answering to evaluating a first-order query over a database which represents the ABox of the knowledge base. Such a property is called *FOL-reducibility* of query answering. Since first-order queries can be expressed in SQL, the importance of FOL-reducibility is that, when query answering enjoys this property, we can take advantage of Data Base Management System (DBMS) techniques for both representing data, i.e., ABox assertions, and answering queries via reformulation into SQL[4].

Turning back to the showed results, we have that for DLs for which query answering is NLogSpace-hard and PTime-hard, query answering is not FOL-reducible, but at least the power of linear recursive Datalog (NLogSpace) and general recursive Datalog (PTime) are required to define a declarative specification of the problem. Note that, although very interesting and promising Datalog engines exist, query optimization strategies for this query language are not sufficiently mature yet to deal with complex applications with millions of instances in the extensional level.

We are currently studying, and will keep on studying in the next months within WP 4, cases for which query answering is FOL-reducible. A first line of research consists in finding out DLs for which answering of expressive queries (e.g., conjunctive queries) is FOL-reducible. A second line of research consists in considering cases which are not FOL-reducible, since the data complexity goes beyond LogSpace or the problem is even undecidable, and interpreting them according to an alternative semantics for query answering, based on the use of an epistemic operator, which weakens the FOL-based semantics of query answering presented in Section 1.3.1. Such a semantics allows us to make use of query languages that are both close in expressive power to FOL, and for which query answering is decidable (and, possibly, FOL-reducible).

2 Query Formulation Support

In the context of access to data sources mediated by ontologies users should be guided towards the precise formulation of their queries, in order to obtain only relevant answers. This process should be supported by automated reasoning tasks which make use of the ontologies describing the data sources.

[4]We consider here the kernel of the SQL-92 standard, i.e., we see SQL as an implementation of relational algebra.

Within this perspective, an "intelligent" query interface supports a user in formulating a precise query – which best captures her/his information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data.

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The ontology defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's rich vocabulary. The user can exploit the ontology's vocabulary to formulate the query, and she/he is guided by such a richer vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of such interface.

Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries.

Queries can be specified through an iterative refinement process supported by the ontology through intensional navigation. The user may specify her/his request using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, giving instead an explicit meaning to a query and to its subparts through classification.

In the literature there are several approaches at providing intelligent visual query systems for relational or object oriented databases (see [22] for an extensive survey). However, to our knowledge, not much has been done in the context of ontology-based query processing (see [23]).

The strength of the presented approach derives from the fact that the graphical representation of the queries is underpinned by a formal semantics provided by an ontology language. The use of an appropriate ontology language enables the system engineers to precisely describe the data sources, and their implicit data constraints, by means of a system global ontology (see [21]). The same ontology is leveraged by the query interface to support the user in the composition of the query, rather than relying on a less expressive logical schema. The underlying technology used by the query interface is based on the recent work on query containment under constraints (see [19, 51]).

In the following sections we describe the underpinning technologies and techniques enabling the query user interface. We will start by describing our assumptions on the query language. The whole system is supported by formally defined reasoning services which are described in Section 2.2.

Since the interface is build around the concept of classes and their properties, we consider conjunctive queries composed by unary (classes) and binary (attribute and associations) terms.

$$\{x_1, \ldots, x_k \mid T_1(y_1), \ldots, T_n(y_n), R_1(z_1, w_1), \ldots, R_\ell(z_\ell, w_\ell)\}$$

where the letters x, y, z, w denote variables or basic data constants (numbers, strings, etc.), T and R are unary and binary terms respectively. Unary terms represent class membership, while binary terms connect classes or datatypes by means of attributes.

Variables $x_1, \ldots, x_k$ are called *distinguished*, and represent the values which are going to be returned by the query. The variables in the *body* of the query $T_1(y_1)$, …, $T_n(y_n)$, $R_1(z_1, w_1)$, …, $R_\ell(z_\ell, w_\ell)$ are considered as existentially quantified. The distinguished variables should be among the variables appearing in the body. The result of a query is the set of k-tuples of values which, substituted to the distinguished variables, make the body of the query satisfied in the integrated view of the data sources.

For example, a query to retrieve the suppliers selling on the Italian market would be

$$\{x \mid \mathrm{Suppl}(x), \mathrm{sell\_on}(x, y), \mathrm{It\_market}(y)\}.$$

The body of a query can be considered as a graph in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) when for the corresponding graph the same property holds. We restrict ourselves to acyclic connected queries. This restriction is dictated by the requirement that the casual user must be comfortable with the language itself.[5] Note that the query language restrictions do not affect the ontology language, where the terms are defined by a different (in our case more expressive) language. The complexity of the ontology language is left completely hidden to the user, who doesn't need to know anything about it. Under this assumption, we do not need to explicitly use variable names since the paths from the root unequivocally individuate each variable.

Let us consider for example the query "Supplier and Multinational corporation located in Europe and selling on Italian market". Firstly, a new variable (x_1) is associated to the top level "Supplier and Multinational corporation". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1 \mid \mathrm{Suppl}(x_1), \mathrm{Mult\_corp}(x_1), \ldots\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "selling on", with its value restriction "Italian market": this introduces a new variable $x_{1,1}$. The second branch is expanded in the same way generating the conjunctive query

$$\{x_1 \mid \mathrm{Suppl}(x_1), \mathrm{Mult\_corp}(x_1), \mathrm{sell\_on}(x_1, x_{1,1}), \\ \mathrm{It\_market}(x_{1,1}), \mathrm{loc\_in}(x_1, x_{1,2}), \mathrm{Eur}(x_{1,2})\}.$$

2.1 Query Building

In this context we are not directly interested in the way the user interacts with the system (see [23] for an example). However, to make the explanation easier to follow, we assume that queries are represented by means of tree diagrams which can be edited by the user. The user can select arbitrary subparts of the actual query and apply appropriate operations which are suggested by the system (e.g. by means of pop-up menus). Query building is performed by means of the query manipulation diagram which enables the user to modify the selected part of the query. With the formalisms introduced in the previous

[5]Our technique can deal with disjunction of conjunctive queries, even with a limited form of negation applied to single terms. See [19, 51] for the technical details.

sections we are now able to give a precise meaning to the selection of sub-parts of the query on the text box.

Since a query is a tree, the focus corresponds to a selected sub-tree. It is easy to realize that each sub-tree is univocally identified by the variable corresponding to a node. Therefore, the focus is always on variable, and moving the focus corresponds to selecting a different variable. Modifying a query sub-part means operating on the corresponding sub-tree modifying the corresponding query tree.

Substitution by navigation corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be a tree composed by a single node, without any branch, whose unary term is the given ontology term.

In the *refinement by compatible terms*, the selected terms are simply added to the root node as unary query terms. For the *property extension*, adding an attribute or associations corresponds to the creation of a new branch. This operation introduces a new variable (i.e. node) with the corresponding restriction. When an attribute is selected, and a constant (or an expression) is specified, then this is added as restriction for the value of the variable.

2.2 Reasoning Services and Query Interface

Reasoning services w.r.t. the ontology are used by the system to drive the query interface. In particular, they are used to discover the terms and properties (with their restrictions) which are proposed to the user to manipulate the query.

Our aim is to be as less restrictive as possible on the requirements for the ontology language. In this way, the same technology can be adopted for different frameworks, while the user is never exposed to the complexity (and peculiarities) of a particular ontology language.

In our context, an ontology is composed by a *set of predicates* (unary, binary), together with a *set of constraints* restricting the set of valid interpretations (i.e. databases) for the predicates. The kind of constraints which can be expressed defines the expressiveness of the ontology language. Note that these assumptions are general enough to take account of widely used modeling formalisms, like UML for example.

We do not impose general restrictions on the expressiveness of the ontology language, however, we require the availability of two *decidable* reasoning services: satisfiability of a *conjunctive query*, and containment test of two conjunctive queries, both w.r.t. the constraints. If the query language includes the *empty* query (i.e. a query whose extension is always empty), then query containment is enough (a query is satisfiable iff it is not contained in the empty query). The query building interface represents the available operations on the query w.r.t. the current focus, i.e. the variable which is currently selected. Therefore, we need a way of describing a conjunctive query from the point of view of a single variable. The expression describing such a viewpoint is still a conjunctive query, which we call *focused*. This new query is equal to the original one, with the exception of the distinguished (i.e. free) variables: the only distinguished variable of the focused query is the variable representing the focus. In the following we represent as q^x the query q

focused on the variable x. For example, the query

$$q \equiv \{x_1, x_{1,2} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \\ \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\},$$

focused in the variable $x_{1,1}$ would simply be

$$q^{x_{1,1}} \equiv \{x_{1,1} \mid \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \\ \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

The operations on the query expression require two different types of information: *hierarchical* (e.g. substitution by navigation), and on *compatibility* (e.g. refinement and new properties).

The first type answers to the question *"Which are the terms more general/more specific/equivalent?"* w.r.t. the focused query, this question can be directly transformed into a containment checking. The second type of information corresponds to a satisfiability check – that is, to verify whether a given term (property) can be added to the query without causing the unsatisfiability of the query itself.

Let us consider the substitution by navigation with the more specific terms (the cases with more general and equivalent terms are analogous). Given the focused query q^x, we are interested in the unary atomic terms T such that the query $\{y \mid T(y)\}$ is contained in q^x and it is most general (i.e. there is no other query of that form contained in q^x, and containing $\{y \mid T(y)\}$).

Refinement by compatible terms and the addition of a new property to the query require the list of terms "compatible" with the given query. In terms of conjunctive queries, this corresponds to add a new term to the query. The term to be added should "join" with the query by means of the focused variable, and must be compatible in the sense that the resulting query should be satisfiable. This leads to the use of satisfiability reasoning service to check which predicates in the ontology are compatible with the current focus. With unary terms this check corresponds simply to adding of the term $T(x)$ to the focused query q^x and verifying that the resulting query is satisfiable.

The addition of a property requires the discovery of both a binary term and its restriction: the terms to be added are of the form $\{x \mid R(x,y), T(y)\}$ if the focused variable is x. As for the refinement by compatible terms, the system should check all the different binary predicates from the ontology for their compatibility. This is practically performed by verifying the satisfiability of the query $q^x \bowtie \{x \mid R(x,y)\}$, for all atomic binary predicates R in the signature and where y is a variable not appearing in q.[6] Once a binary predicate R is found to be compatible with the focused query, the restriction is selected as the most general unary predicate T such that the query $q^x \bowtie \{x \mid R(x,y), T(y)\}$ is satisfiable.

If such a predicate does not exist (e.g. there are two predicates incomparable w.r.t. the constraints) then the property is not proposed to the user. The reason for this choice is that the ontology does not provide enough specification for the restriction in this context;

[6]Here $\bowtie$ represents a natural join.

so, if the ontology reflects the underlying data, its presence in the query would not be a significant discrimination to find the results the user is looking for.

This choice, which can be easily relaxed, seems to provide a good heuristic for most of the cases. However, tests with users and different ontologies are required to really understand if such kind of restrictions are satisfactory when the framework is applied to different domains or user typologies.

2.3 Requirements for Ontologies

The whole interface is driven by algorithms which make an extensive use of logical reasoning services on top of the ontology. This means that the effectiveness of the interface depends almost entirely on the quality of the ontology (see [29]). In particular, a poor ontology will result in the interface being cluttered by irrelevant terms and properties. This will confuse the user, which would not receive a good support from the system in the refinement of the query.

The appropriate use of disjointness and covering constraints in the ontology is crucial for the system being able to prune insignificant terms from the query manipulation diagram. Lacks of relevant disjointness constraints (w.r.t. the modelled domain) affects the detection of relevant properties, and compatible terms of the selected sub-query. This leaves the user in the position of relying only on the taxonomical structure of the ontology terms.

To understand the reasons behind the importance of disjointness constraints, the reader should consider that in principle any terms and property is compatible with any other term or property unless otherwise stated in the ontology. For example, the user would find terms like “Multinational corporation” among the terms compatible with “Student”, or attributes like “city” applicable to “Trousers”. An ontology without disjointness constraint would cause the interface to propose any attribute in the ontology as an applicable property to any term. This is obviously meaningless to the user, which would be forced to guess the right properties without any support. Bear in mind, that adding a property means restricting the result to URIs having the property, therefore, adding the wrong attribute would return no result at all.

Although not as crucial as disjointness, covering constraints provide an extremely powerful tool for discovering equivalences and properties using reasoning by cases.[7] Therefore, it enhances the user experience with the interface, as the system is able to pinpoint precise possible refinements for the sub-query.

2.4 Beyond Binary Predicates

Up until now we considered binary predicates only, but in general an ontology may contains n-ary predicates. The user should be provided with a means of querying databases modelled by such ontologies, however, the query building interface has to present a uniform approach to query composition. For this reason, for the purpose of query design, n-ary predicates are reified by using binary predicates representing the projection of the tuples in their positional components. The user composes the queries in the same way as

[7]When used in conjunction with disjointness.

described in the previous sections; at the end, before being sent to the query evaluator, the conjunctive query is transformed by collapsing the reified binary components into the appropriate n-ary predicates.

The reification introduces a new variable representing an n-ary tuple. This variable can be seen as the primary key for the tuple, or as an object identifier if we refer to object oriented data models. Note that since these variables do not appear in the query sent to the evaluator, there are no problems concerning the same tuple appearing more than once with different identifiers (moreover, we do not require any change in the data model of the ontology).

For example, the ternary predicate Employee relating an employee with a company and her/his role is reified by using the projection predicates π_1, π_2, π_3. The query term $\text{Employee}(x, y, z)$ corresponds to the reified terms $\text{Employee}(\mathbf{t}), \pi_1(\mathbf{t}, x), \pi_2(\mathbf{t}, y), \pi_3(\mathbf{t}, z)$. Since the system knows the arity of the predicates, it does not need all variables for each tuple, which can be added if necessary. For example the terms $\text{Employee}(\mathbf{t}), \pi_2(\mathbf{t}, x)$ can be transformed into $\text{Employee}(x', x, x'')$ without the user intervention.

Given the fact that tuple-representing variables disappear at the evaluation time, the user is not allowed to select them as distinguished variables. For this reason we provide a simplified query building mode in which the tuple-representing variables are never exposed to the user. In this mode binary predicates corresponding to the composition of two projections are presented to the user. For example the $\text{Works\_for}(x, y)$ predicate corresponds to the query terms $\text{Employee}(\mathbf{t}), \pi_1(\mathbf{t}, x), \pi_2(\mathbf{t}, y)$.

Note that, by hiding the tuple representing variables, the user has a restricted query language expressiveness. In particular it is not possible to impose that more than two variables participate in the same tuple. Let us consider the Employee predicate, with the binary "shortcuts" Works_for, and Has_role connecting the first to the third components. The query containing the terms $\text{Works\_for}(x, y), \text{Has\_role}(x, z)$ does not enforce x, y, z to belong to the same tuple.

2.5 Using a Description Logics Reasoner

Although our approach is not tight to any ontology language, in the test implementation of our system we are using Description Logics (DLs). The reasons for this choice lie in the facts that DLs can capture a wide range of widespread modelling frameworks, and the availability of efficient and complete DL reasoners.

We adopted the Description Logics $\mathcal{SHIQ}$ (see [50]) which is expressive enough for our purposes, and for which there are state of the art reasoners. Note that the adoption of $\mathcal{SHIQ}$ allows us to use ontologies written in standard Web Ontology languages like OWL–DL (see [49]). One of the key features of $\mathcal{SHIQ}$ is the possibility of expressing the inverse of a role which is extremely useful for converting tree–shaped queries into DL concept expressions.

Given the restriction to tree–shaped conjunctive query expressions, together with the availability of inverse roles, a focused query (see Section 2.2) corresponds to a concept expression (see [53]). Therefore, all the reasoning tasks described in Section 2.2 correspond to standard DL reasoning services. Again, this is not a restriction imposed by the underlying technology, since general conjunctive queries can be dealt with techniques described in [19, 51].

The idea behind the transformation of a query expression into a single concept description is very simple, and it is based on the fact that a concept expression can be seen as a query with a single distinguished variable. To focus the query on a variable, we start from the variable itself, then we traverse the query graph by encoding binary terms into DL existential restrictions and dropping the variable names. The fact that queries are tree–shaped ensures that variable names can be safely ignored. Let us consider for example the query expression

$$\{\text{Mult\_corp}(x_1), \text{Italian}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\}.$$

The DL expression corresponding to the query focused on $x_{1,1}$ is

$$(\text{It\_market} \sqcap \exists \text{sell\_on}^{-}.(\text{Mult\_corp} \sqcap \text{Italian}))$$

where sell_on^{-} corresponds to the inverse of sell_on role.

As explained in Section 2.2, we need two kinds of information: compatibility and hierarchical. These, in the DL framework, are provided by the standard reasoning services of satisfiability and taxonomy position of a concept expression respectively. The first service verifies the satisfiability w.r.t. a knowledge base, while the second classifies a concept expression (i.e., provides it w.r.t. the ISA taxonomy of concept names). [8] Reasoning tasks described in Section 2.2 can be straightforwardly mapped into satisfiability and classification.

For example, checking the compatibility of the term Italian with the query

$$\{\text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\},$$

is performed by checking the satisfiability of the concept

$$\text{Italian} \sqcap \text{Mult\_corp} \sqcap \exists \text{sell\_on}.\text{It\_market}.$$

Compatibility of binary terms is performed analogously by using an existential restriction, e.g., $\exists$sell_on.$\top$.[9] To discover the restriction of a property we use classification instead of repeated satisfiability. The idea is to classify the query focused on the variable introduced by the property. For example, to discover the restriction of sell_on applied to the query expression

$$\{x_1 \mid \text{Mult\_corp}(x_1), \text{Italian}(x_1)\},$$

we classify the expression $\exists$sell_on^{-}.(Mult_corp $\sqcap$ Italian). The DL reasoner returns the list of concept names more general and equivalent to the range of the relation sell_on, when restricted to the domain (Mult_corp $\sqcap$ Italian). This is exactly the information we need to discover the least general predicate(s) which can be applied to the property in the given context.

Our implementation uses the DL reasoner Racer (see [39]) which fully supports the $\mathcal{SHIQ}$ DL. The interaction with the DL reasoner is based on the DIG 1.0 interface API (see [12]), a standard to communicate with DL reasoners developed by different DL systems implementors. This choice makes our system independent from a particular DL reasoner, which can be substituted with any DIG based one.

[8] DL systems usually provide an efficient way of obtaining the taxonomic position of a given concept expression.

[9] Note the use of the $\top$ concept representing the whole domain (any possible concept).

3 Information Extraction

Query answering w.r.t. ontologies requires that explicit descriptions are available. In one of the standard ontology settings with description logics these explicit descriptions are available as an ABox. However, it is not always possible to easily transform a given information source into an ABox. In particular, if the information source contains media data such as still images, audio and video files, or natural language text, it is an open research problem how to automatically represent media content. Without loss of generality, in the following, we consider only (still) images. In this case, it is assumed that information objects, e.g., a particular image, is annotated with so-called meta data that can be seen as (part of) an ABox. If meta data is not available, in order to support ontology-based query answering, meta data must be derived automatically. Thus, an image interpretation problem has to be solved. One approach is to use ontologies in general, and description logics in particular. Examining image interpretation tasks will lead to insights about how to organize ontology access, processing, and usage since specific patterns of inference service calls will emerge.

3.1 Formalizing the Information Extraction Problem

A formalization of scene interpretation as abduction is presented by Shanahan in [82]. Shanahan defines scene interpretation using the entailment decision problem in an abductive context. Based on a background knowledge base Σ, the task is to compute formulas Δ such that formulas Γ, which represent visual percepts of an agent, are entailed: $\Sigma \cup \Delta \models \Gamma$.[10] In other words: Δ is to be abductively determined such that all models of $\Sigma \cup \Delta$ are also models of Γ. The computed formulas Δ can be seen as an explanation for the percepts Γ. Note that in this approach the formulas Γ are defined by external processes not covered here. Δ need not be empty initially, thus Δ might contain some facts different from Γ that are taken for granted and need not be explained but may be used to explain Γ. In Shanahan's approach, the scene interpretation problem is solved if some set of formulas Δ satisfying $\Sigma \cup \Delta \models \Gamma$ is computed. However, there are certain characteristics that the explanation Δ should fulfill in order to give a useful explanation:

- $\Sigma \cup \Delta$ must be satisfiable.
- $\Gamma \not\subseteq \Delta$: It is of course true that an entailment such as $A \models A$ is true, but having an explanation similar to the observations for the sake of fulfilling the entailment problem, is not a solution. Having a statement such as *The sky is blue because the sky is blue* is not of much help. Thus, it is important that the explanation Δ does not (syntactically) contain the facts Γ.
- Δ should be determined such that there is no Δ' such that also $\Sigma \cup \Delta' \models \Gamma$ and $\Delta \models \Delta'$ holds, i.e., the best explanation is the one imposing as few restrictions as possible (minimality condition).

[10] We slightly simplify the approach of [82] in order to emphasize the main ideas.

Depending on the application context, Shanahan also considers actions of an agent. In order to model actions in a first-order context, he uses circumscription [80, 358ff.] to cope with the frame problem. Circumscription introduces abnormability terms, which means that many inferences are lost. As usual, the idea is to consider only models that minimize the abnormability terms in the entailment relation. Thus, we have: $\Sigma \cup \Delta \models_{min\_abnorm} \Gamma$.

The approach of [82] can be extended to cope with additional problems in image interpretation. If Δ has been computed, one might extend the percept representation Γ by another part γ which model the expected scene content not yet discovered. The γ represents the expected scene content due to high-level reasoning based on the knowledge base Σ: $\Sigma \cup \Delta \models \Gamma \cup \gamma$. If Δ cannot be found, one might reduce Γ by some δ: $\Sigma \cup \Delta \models \Gamma \setminus \delta$. In the general case something is dropped while some other parts are expected due to high-level reasoning: $\Sigma \cup \Delta \models (\Gamma \setminus \delta) \cup \gamma$. In all cases, some set of assertions Δ satisfying the above-mentioned conditions has to be determined.

Before we analyze how the derivation of a suitable Δ can be supported by adequate ontology processing technologies, let us have a look at the example shown in Figure 1.

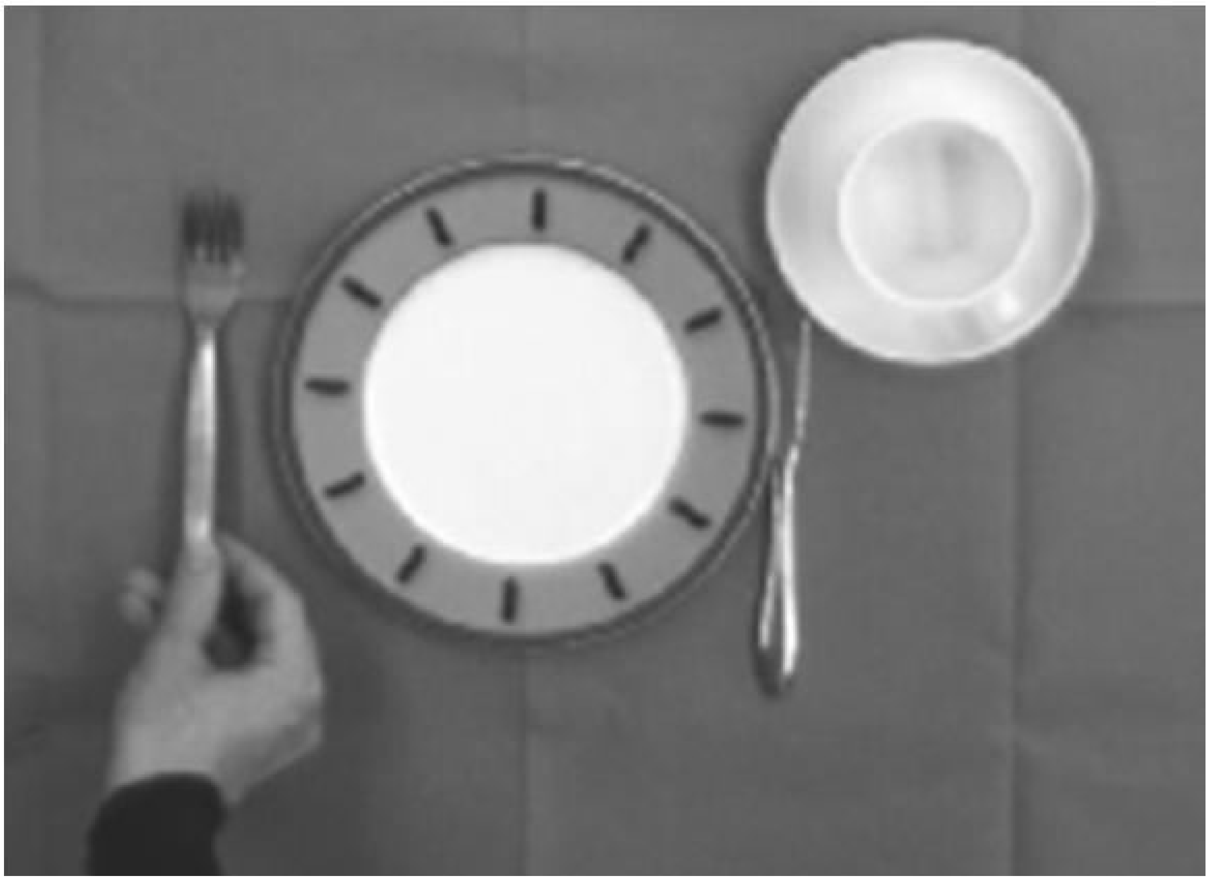

Figure 1: What is the interpretation of this image?

We assume that the following can be detected by standard image analysis techniques:

1. A set of objects in the image, e.g., a saucer, a plate, a fork and a knife (maybe the cup on the saucer is not recognized initially).

2. A spatial configuration of the elements, i.e., the fork is located to the left of the plate, the knife is located to the right and so on.

If we were to interpret this image, then a reasonable explanation of the spatial configuration of the elements should be found. In other words, there is the assumption that the spatial configuration of the objects in the image is not random, but it rather has a purpose. Thus, the interpretation should explain such spatial configuration between the objects. We can model the spatial configuration of the objects as a set of assertions Γ: $\{\mathit{left\_of}(\mathit{fork_1}, \mathit{plate_1}), \mathit{right\_of}(\mathit{knife_1}, \mathit{plate_1}), \ldots\}$.

We assume background knowledge about covers is represented as part of Σ. Thus, we assume that Σ contains a representation about the parts of a cover, their relations, etc. represented in an appropriate language (e.g., description logics with rules). Then, in order to solve the equation $\Sigma \cup \Delta \models \Gamma$, a cover instance must be added to Δ with $fork_1, plate_1, \ldots$ being the parts. Then, with an appropriate definition of the concept $Cover$ in Σ, Γ will be entailed. In order to keep the discussion brief, we do not present all details about how to represent the required knowledge (see [72] for an initial approach).

In this way, we have formalized the interpretation problem as a logical decision problem, namely the entailment problem. The process for computing a suitable Δ will involve a search process with multiple calls to standard description logic inference services. For natural language interpretation similar insights can be gained from, e.g., [30, 15].

In this section we have considered "syntactic" additions (and implicitly retractions) of assertions to ABoxes as part of certain incremental high-level reasoning processes. In Section 4.3, the issue of updates to ABoxes will be discussed from a semantic point of view.

3.2 Challenges for Ontology Access, Processing, and Usage

The formalization of the image interpretation task as a logical decision problem is an example for the use of ontologies for information management tasks, in this case the automatic computation of meta data for, e.g., sets of media objects. In this example scenario, the underlying task-specific search process involves solving multiple entailment problems, and maybe invoking several other standard inference services provided by ontology systems.

The entailment problem can be reduced to other decision problems: It holds that $\Sigma \cup \Delta \models \Gamma$ if $(\Sigma \wedge \Delta) \rightarrow \Gamma$ is a tautology, or the negation is not satisfiable: $\neg SAT(\neg((\Sigma \wedge \Delta) \rightarrow \Gamma))$.

During the construction process for Δ, the set of assertions will be incrementally extended. After each extension, one of the above-mentioned entailment decision problems is solved. In order to make the use of ontologies in image interpretation practical, ontology processing systems (ontology servers) must support reasoning processes that can adequately cope with incremental changes to ABoxes that are managed by the systems. In some DL systems, only reference implementation for changes to ABoxes managed by ontology servers are available (e.g., in Racer) while in others (e.g., Pellet) initial experiments for incremental reasoning have been carried out [46]. New research is required to efficiently solve incremental satisfiability problems or even incrementally answer queries.

4 Updating ABoxes

4.1 Motivation

Changes to ontologies occur at design time (offline time) as well as at usage time (online time). Except for some online learning scenarios, if ontologies based on description logics are considered, usually, design-time changes mostly concern changes to the intensional part (TBox) whereas usage-time changes affect the extensional part (ABox). The growing acceptance of ontology-based methods and technologies in practical applications, in particular in the context of Semantic Web, revealed new challenges for state-of-the-art DL systems resulting from the fact that knowledge bases now have in general a dynamic content. There are a number of application scenarios where knowledge is frequently updated on the conceptual level or/and on the extensional level. The task of media interpretation discussed in the previous section is one example for such use cases. Latest investigations on knowledge base updates pursue two different directions. While some approaches study so-called syntactic updates and incremental reasoning (see [46, 45, 40]), others make an attempt to define a formal update semantics based on model theory rather than told information ([33, 64, 79, 77, 44]). In what follows, we give a short overview of research on both update approaches, placing emphasis on updating ABoxes.

4.2 Syntactic Updates

Updating ABoxes syntactically means that just syntactic assertions (concept assertions, role assertions, or concrete domain assertions) are explicitly added to or removed from an ABox. Restricting updates to be only a syntactic operation implies there is no warranty that the a removed assertion will not be entailed anymore. Analogously, it is not guaranteed that after adding a new assertion to an ABox the knowledge base will be still consistent. A syntactic update is defined in terms of set operations. Assuming S to be a set of assertions in an initial ABox, $S' = S \cup \alpha$ denotes the result of adding an axiom α to S and $S'' = S \setminus \alpha$ denotes the result of removing an axiom α from S.

Apart from the fact that in case of a syntactic approach the above-mentioned semantic side effects of updates are simply ignored, contemporary highly-optimized sound and complete DL reasoners still show a lack of performance when dealing with incremental updates. While in some former incomplete systems, the topic of incremental reasoning was already an issue (e.g., [65]), initially, sound and complete DL systems provided support for incremental addition and retraction of ABoxes assertions only as reference implementations (e.g., Racer [39]). The interfaces were there, but internal data structures used for answering queries were just recomputed from scratch, i.e., nothing what has been computed before was exploited after some assertions were added. Obviously, this caused performance problems. Therefore, optimizing techniques for incremental reasoning and query answering are crucial for modern sound and complete DL systems. First investigations for reasoning algorithms considering incremental ABox updates are presented in [46]. However, since research on efficient query answering techniques w.r.t. expressive description logics has just begun, support for syntactic updates is just in its infancy.

4.3 Semantic Updates

Syntactically retracting an assertion does not mean that it is no longer entailed. In other words, in order to make sure an assertion does not hold, one has to identify a set of assertions such that if one of the assertions is retracted, a certain conclusion is no longer valid. On the other hand, one might argue that adding an assertion to an ABox might involve "weakening" other assertions (maybe for particular individuals) such that an inconsistency can be avoided. In order to accomplish this, a formal approach is required. Updates become reasoning problems.

Formal theory of ABox updates essentially utilizes a possible model approach, initially elaborated in the context of reasoning about actions [90]. In accordance with this approach, the following view is appropriate. An ABox represents the current state-of-affairs in the application domain. Since the description of the world provided by the ABox is usually indefinite, an ABox has multiple models. In case of an update, it is unknown, which of the possible models describes the actual state-of-affairs. Therefore updates must refer to all possible models. The result is a new set of models representing the updated state of the domain. The task is to determine minimal changes, i.e., changes absolutely required in the actual model for the state-of-affairs. Here, the aim is to retain all knowledge that is not affected by the change.

Actually, two recent approaches have to be developed in the context of description logics as part of the TONES [33, 64]. Both research contributions adapt the possible model semantics of updates (details can be found, e.g., in [34, 90]) to provide a general notion of update in absence for description logic ABoxes. Following the definitions in [33], we assume an (initial) knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a finite set of (new) assertions $\mathcal{F}$ such that $\mathcal{F}$ is consistent w.r.t. $\mathcal{T}$ (that is $Mod(\mathcal{T}) \cap Mod(\mathcal{F}) \neq \emptyset$, where $Mod(\mathcal{T})$ denotes the set of models of $\mathcal{T}$ and $Mod(\mathcal{F})$ denotes the set of models of $\mathcal{F}$). The result of updating of $\mathcal{A}$ with $\mathcal{F}$ w.r.t $\mathcal{T}$ is the union of updates of models of $\mathcal{K}$ with $\mathcal{F}$, written as $\bigcup_{\mathcal{I} \in Mod(\mathcal{K})} U^{\mathcal{T}}(\mathcal{I}, \mathcal{F})$, where $U^{\mathcal{T}}(\mathcal{I}, \mathcal{F})$ stands for the result of updating $\mathcal{I}$ with $\mathcal{F}$ and is defined as follows:

$$U^{\mathcal{T}}(\mathcal{I}, \mathcal{F}) = \{\mathcal{I}' \mid \mathcal{I}' \in Mod(\mathcal{T}) \cap Mod(\mathcal{F}) \text{ and there exists no } \mathcal{I}'' \in Mod(\mathcal{T}) \cap Mod(\mathcal{F}) \text{ such that } ((\mathcal{I} \cup \mathcal{I}'') \setminus (\mathcal{I} \cap \mathcal{I}'')) \subset ((\mathcal{I} \cup \mathcal{I}') \setminus (\mathcal{I} \cap \mathcal{I}'))\}.$$

In this definition, updates are based on models. Therefore, these updates are called "semantic" updates. In [64], updating ABoxes in several expressive DLs is studied. It is shown that, given the formal definition of semantic updates above, in many DLs, updated ABoxes are no longer expressible in the language of the initial knowledge base even if TBoxes are not considered. It turned out that DLs have to include nominals and the "@" constructor of hybrid logic (or, equivalently, admit Boolean ABoxes) for updated ABoxes to be expressible. The authors emphasize that an important issue is the size of the updated ABoxes. Indeed, updates can lead to an exponential blowup in the size of the original ABoxes. Although for the logic $\mathcal{DL}{-}Lite$ the result of an update is proved to be always expressible in the logic of the initial knowledge base, and an algorithm that computes the update over a $\mathcal{DL}{-}Lite$ knowledge base runs in polynomial time w.r.t. the size of the original knowledge base (see [33] for details), in general, the computation of updates is a non-trivial task.

Another challenge concerns conditional updates which are useful in some applications (e.g., reasoning about actions [10]). Conditional updates are investigated in [64].

Up to now, updates at the instance level have been seen as the task of handling the situations in which only extensional information changes. However, updates may be propagated from the conceptual layer to the data layer or vice versa causing inconsistencies between update information and the intensional level of the original ontology. This observation made in the literature about updates gives rise for further studies in the context of ontologies.

Part II
Using Ontologies for Specific Tasks in Applications

5 Semantic Service Discovery and Selection

Description logics play an important rôle in the Semantic Web since they are the basis of the W3C-recommended Web ontology language OWL [8] ,which can be used to create semantic annotations describing the content of Web pages. In addition to static information, the Web also offers services, which allow their users to effect changes in the world, like buying a book or opening a bank account. As in the case of static information, annotations describing the semantics of the service should facilitate discovery of the right service for a given task. Since services create changes of the world, a faithful representation of its functionality should deal with this dynamic aspect in an appropriate way.

In AI, the notion of an action is used both in the planning and the reasoning about action communities to denote an entity whose execution (by some agent) causes changes of the world (see e.g. [78]). Thus, it is not surprising that theories developed in these communities have been applied in the context of Semantic Web services. For example, [69] use the situation calculus [78] and GOLOG [59] to formalize the dynamic aspects of Web services and to describe their composition.

In this project the focus is on the faithful description of the changes to the world induced by the invocation of a service. To this purpose, services are described as actions that have pre-conditions and post-conditions (its effects). These conditions are expressed with the help of DL assertions, and the current state of the world is (incompletely) described using a set of such assertions (an ABox). In addition to atomic services, we also consider simple composite services, which are sequences of atomic services. The semantics of a service is defined using the possible models approach developed in the reasoning about action community [90] and is fully compatible with the usual DL semantics. However, it has been shown in [9, 11] that this semantics can be viewed as an instance of Reiter's approach [78] for taming the situation calculus. In particular, our semantics solves the frame problem in precisely the same way.

We concentrate on two basic reasoning problems for (possibly composite) services: executability and projection. *Executability* checks whether, given our current and possibly incomplete knowledge of the world, we can be sure that the service is executable, i.e., all pre-conditions are satisfied. *Projection* checks whether a certain condition always holds after the successful execution of the service, given our knowledge of the current state of the world. Both tasks are relevant for service discovery. It is obviously preferable to choose a service that is guaranteed to be executable in the current (maybe incompletely known) situation. In addition, we execute the service to reach some goal, and we only want to use services that achieve this goal. Though these reasoning tasks may not solve the discovery problem completely, they appear to be indispensable subtasks.

The main aim of this work is to show how executability and projection can be com-

Symbol	Constructor	$\mathcal{ALC}$	$\mathcal{ALCO}$	$\mathcal{ALCQ}$	$\mathcal{ALCI}$	$\mathcal{ALCQO}$	$\mathcal{ALCIO}$	$\mathcal{ALCQI}$
$\mathcal{Q}$	$(\leqslant n\ r\ C)$ $(\geqslant n\ r\ C)$			x		x		x
$\mathcal{I}$	r^-				x		x	x
$\mathcal{O}$	$\{a\}$		x			x	x	

Figure 2: Fragments of $\mathcal{ALCQIO}$.

puted, and how their complexity depends on the description logic used within our framework. For the DLs $\mathcal{L}$ considered here, which are all sublanguages of the DL $\mathcal{ALCQIO}$, the complexity of executability and projection for services expressed in this DL coincides with the complexity of standard DL reasoning in $\mathcal{L}$ extended with so-called nominals (i.e., singleton concepts).

5.1 The Formalism

The framework for reasoning about Web services is not restricted to a particular DL, but can be instantiated with any DL that seems appropriate for the application domain at hand. Most complexity results were established for the DL $\mathcal{ALCQIO}$ and a number of its sublanguages [11, 10, 9], which form the core of OWL-DL. The additional OWL-DL constructors could be easily added, with the exception of transitive roles.

We recall the constructors available in $\mathcal{ALCQIO}$ and its fragments, which are obtained by omitting some constructors.

Definition 25 ($\mathcal{ALCQIO}$ Syntax)**.** Let $\mathsf{N_C}$ be a set of *concept names.* A *role* is either a role name or the inverse r^- of a role name r. The set of $\mathcal{ALCQIO}$-concepts is the smallest set satisfying the following properties: (1) each concept name $A \in \mathsf{N_C}$ is a concept and (2) if C and D are concepts, r is a role, a an individual name, and n a natural number, then the following are also concepts:

$\neg C$	(*negation*)	$\{a\}$	(*nominal*)
$C \sqcap D$	(*conjunction*)	$(\geqslant n\ r\ C)$	(*atmost number restriction*)
$C \sqcup D$	(*disjunction*)	$(\leqslant n\ r\ C)$	(*atleast number restriction*)
$\top$	(*top*)	$\exists R.C$	(*existential restriction*)
$\bot$	(*bottom*)	$\forall R.C$	(*universal restriction*)

△

Names and concept constructors for different fragments of $\mathcal{ALCQIO}$ considered in our framework are shown in Figure 2. We now define the semantics of $\mathcal{ALCQIO}$ concepts.

Definition 26 ($\mathcal{ALCQIO}$ Semantics)**.** An interpretation $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a mapping that assigns to each concept name A, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to each individual name a, an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$; and to each role name r, a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

The interpretation of inverse roles and complex concepts is then defined as follows, with $\#S$ denoting the cardinality of the set S:

$$\begin{array}{rcl@{\qquad}rcl}
(\{a\})^{\mathcal{I}} & = & \{a^{\mathcal{I}}\} & (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} & (\leqslant n\ r\ C)^{\mathcal{I}} & = & \{d \mid \#\{e \in C^{\mathcal{I}} \mid (d)e \in r^{\mathcal{I}}\} \leq n\} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} & (\geqslant n\ r\ C)^{\mathcal{I}} & = & \{d \mid \#\{e \in C^{\mathcal{I}} \mid (d)e \in r^{\mathcal{I}}\} \geq n\} \\
(r^{-})^{\mathcal{I}} & = & \{(e)d \mid (d)e \in r^{\mathcal{I}}\} & & &
\end{array}$$

An interpretation $\mathcal{I}$ is called a *model* of a concept C if $C^{\mathcal{I}} \neq \emptyset$ △

A central idea of the Semantic Web is to represent relevant terminological knowledge in *ontologies*.

In the DL world, ontologies are usually called TBoxes. A *TBox* is a finite set of concept definitions with unique left-hand sides. We say that a concept name A *directly uses* a concept name B w.r.t. $\mathcal{T}$ if there is a concept definition $A \equiv C \in \mathcal{T}$ with B occurring in C. Let *uses* be the transitive closure of directly uses. Then a TBox $\mathcal{T}$ is *acyclic* if no concept name uses itself w.r.t. $\mathcal{T}$.

For the remainder of this section, we will restrict ourselves to acyclic TBoxes. The reason for this restriction is that cyclic TBoxes cause semantic problems.

To predict the outcome of applying a service, an agent usually needs to take into account her knowledge about the current state of the world – in DLs knowledge about the world is represented in an ABox.

Definition 27 (ABox)**.** An *assertion* is of the form $C(a)$, $r(a,b)$ or $\neg r(a,b)$, where $a, b \in \mathsf{N_I}$, C is a concept, and r a role. An *ABox* is a finite set of assertions. An interpretation $\mathcal{I}$ *satisfies* an assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$; $r(a,b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$; $\neg r(a,b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is called a *model* of an ABox $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, if $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$ △

Negated role assertions are usually not considered in DL, but they are very useful as pre- and post-conditions. As described in [11] reasoning with such assertions can easily be reduced to reasoning without them if the DL under consideration allows for value restriction and atomic negation. The reasoning problem ABox consequence will play an important role in the approach.

Definition 28 (ABox consequence)**.** Let C be a concept, $\mathcal{A}$ an ABox, and $\mathcal{T}$ a TBox. Then an ABox assertion φ is a *consequence* of an ABox $\mathcal{A}$ w.r.t. a TBox $\mathcal{T}$ (written $\mathcal{A}, \mathcal{T} \models \varphi$) if every model of $\mathcal{A}$ and $\mathcal{T}$ satisfies φ. △

It has been shown in [11] that ABox consequence with negated role assertions, i.e. assertions of the form $\neg r(a,b)$, can be polynomially reduced to ABox consistency without negated role assertions, and vice versa.

5.2 Service Descriptions

The formalism for the representation of and reasoning about Web services concentrates on *ground services*, i.e., services where the input parameters have already been instantiated by individual names. *Parametric* services, which contain variables in place of individual names, should be viewed as a compact representation of all its ground instances: a parametric service simply represents the set of all ground services obtained from the parametric service by replacing variables with individual names. The handling of such parametric services takes place "outside" of this formalism and parametric services have already been instantiated. For other tasks, such as planning, it may be more natural to work directly with parametric services.

Definition 29 (Service)**.** Let $\mathcal{T}$ be an acyclic TBox. An *atomic service* $S = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ for an acyclic TBox $\mathcal{T}$ consists of

- a finite set pre of ABox assertions, the *pre-conditions*;
- a finite set occ of *occlusions* of the form $A(a)$ or $r(a, b)$, with A a primitive concept name w.r.t. $\mathcal{T}$, r a role name, and $a, b \in \mathsf{N_I}$;
- a finite set post of *conditional post-conditions* of the form φ/ψ, where φ is an ABox assertion and ψ is a *primitive literal for* $\mathcal{T}$, i.e., an ABox assertion $A(a)$, $\neg A(a)$, $s(a, b)$, or $\neg s(a, b)$ with A a primitive concept name in $\mathcal{T}$ and s a role name.

A *composite service* for $\mathcal{T}$ is a finite sequence $S_1, \ldots, S_k$ of atomic services for $\mathcal{T}$. A *service* is a composite or an atomic service. △

Intuitively, the pre-conditions specify under which conditions the service is applicable. The conditional post-conditions φ/ψ say that, if φ is true before executing the service, then ψ should be true afterwards. The rôle of occlusions is to describe those primitive literals to which the minimization condition does not apply. By the law of inertia, only those facts that are forced to change by the post-conditions should be changed by applying the service.

We can define how the application of an atomic service changes the world, i.e., how it transforms a given interpretation $\mathcal{I}$ into a new one $\mathcal{I}'$, following the *possible models approach* (PMA) initially proposed in [90].

The idea underlying PMA is that the interpretation of atomic concepts and roles should change as little as possible while still making the post-conditions true. Since the interpretation of defined concepts is uniquely determined by the interpretation of primitive concepts and role names, it is sufficient to impose this minimization of change condition on primitive concepts and roles names. We assume that neither the interpretation domain nor the interpretation of individual names is changed by the application of a service.

Formally, we define a precedence relation $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$ on interpretations, which characterizes their "proximity" to a given interpretation $\mathcal{I}$. We use $M_1 \triangledown M_2$ to denote the symmetric difference between the sets M_1 and M_2.

Definition 30 (Preferred Interpretations)**.** Let $\mathcal{T}$ be an acyclic TBox, $S = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ a service for $\mathcal{T}$, and $\mathcal{I}$ a model of $\mathcal{T}$. We define the binary relation $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$ on models of $\mathcal{T}$ by setting $\mathcal{I}' \preccurlyeq_{\mathcal{I},S,\mathcal{T}} \mathcal{I}''$ iff $((A^{\mathcal{I}} \triangledown A^{\mathcal{I}'}) \setminus \{a^{\mathcal{I}} \mid A(a) \in \mathsf{occ}\}) \subseteq A^{\mathcal{I}} \triangledown A^{\mathcal{I}''}$; and $((s^{\mathcal{I}} \triangledown s^{\mathcal{I}'}) \setminus \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid s(a, b) \in \mathsf{occ}\}) \subseteq s^{\mathcal{I}} \triangledown s^{\mathcal{I}''}$. for all primitive concepts A, all role names s, and all domain elements $d, e \in \Delta^{\mathcal{I}}$. When $\mathcal{T}$ is empty, we write $\preceq_{\mathcal{I},S}$ instead of $\preceq_{\mathcal{I},S,\emptyset}$. △

Intuitively, applying the service S transforms the interpretation $\mathcal{I}$ into the interpretation $\mathcal{I}'$ if $\mathcal{I}'$ satisfies the post-conditions and is closest to $\mathcal{I}$ (as expressed by $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$) among all interpretations satisfying the post-conditions. Since we consider *conditional* post-conditions, defining when they are satisfied actually involves both $\mathcal{I}$ and $\mathcal{I}'$. We say that the pair of interpretations $\mathcal{I}, \mathcal{I}'$ *satisfies the set of post-conditions* post ($\mathcal{I}, \mathcal{I}' \models \mathsf{post}$) iff the following holds for all post-conditions φ/ψ in post: $\mathcal{I}' \models \psi$ whenever $\mathcal{I} \models \varphi$.

Definition 31 (Service Application)**.** Let $\mathcal{T}$ be an acyclic TBox, $S = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ a service for $\mathcal{T}$, and $\mathcal{I}, \mathcal{I}'$ models of $\mathcal{T}$ sharing the same domain and interpretation of all individual names. Then *S may transform $\mathcal{I}$ to $\mathcal{I}'$* ($\mathcal{I} \Rightarrow_S^{\mathcal{T}} \mathcal{I}'$) iff (1) $\mathcal{I}, \mathcal{I}' \models \mathsf{post}$, and (2) there does not exist a model $\mathcal{J}$ of $\mathcal{T}$ such that $\mathcal{I}, \mathcal{J} \models \mathsf{post}$, $\mathcal{J} \neq \mathcal{I}'$, and $\mathcal{J} \preccurlyeq_{\mathcal{I},S,\mathcal{T}} \mathcal{I}'$.

The composite service $S_1 \ldots, S_k$ *may transform $\mathcal{I}$ to $\mathcal{I}'$* ($\mathcal{I} \Rightarrow_{S_1,\ldots,S_k}^{\mathcal{T}} \mathcal{I}'$) iff there are models $\mathcal{I}_0, \ldots, \mathcal{I}_k$ of $\mathcal{T}$ with $\mathcal{I} = \mathcal{I}_0$, $\mathcal{I}' = \mathcal{I}_k$, and $\mathcal{I}_{i-1} \Rightarrow_{S_i}^{\mathcal{T}} \mathcal{I}_i$ for $1 \leq i \leq k$. If $\mathcal{T}$ is empty, we write $\Rightarrow_{S_1,\ldots,S_k}$ instead of $\Rightarrow_{S_1,\ldots,S_k}^{\mathcal{T}}$. △

Because of our restriction to acyclic TBoxes and primitive literals in the consequence part of post-conditions, services without occlusions are *deterministic*, i.e., for any model $\mathcal{I}$ of $\mathcal{T}$ there exists at most one model $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow_S^{\mathcal{T}} \mathcal{I}'$. Note that there are indeed cases where there is no successor model $\mathcal{I}'$.

5.3 Reasoning about Services

Assume that we want to apply a composite service $S_1, \ldots, S_k$ for the acyclic TBox $\mathcal{T}$. Usually, we do not have complete information about the world (i.e., the model $\mathcal{I}$ of $\mathcal{T}$ is not known completely). All we know are some facts about this world, i.e., we have an ABox $\mathcal{A}$, and all models of $\mathcal{A}$ together with $\mathcal{T}$ are considered to be possible states of the world. Before trying to apply the service, we want to know whether it is indeed executable, i.e., whether all necessary pre-conditions are satisfied. If the service is executable, we may want to know whether applying it achieves the desired effect, i.e., whether an assertion that we want to make true really holds after executing the service. These problems are basic inference problems considered in the reasoning about action community, see e.g. [78]. In our setting, they can formally be defined as follows:

Definition 32 (Reasoning Services)**.** Let $\mathcal{T}$ be an acyclic TBox, $S_1, \ldots, S_k$ a service for $\mathcal{T}$ with $S_i = (\mathsf{pre}_i, \mathsf{occ}_i, \mathsf{post}_i)$, and $\mathcal{A}$ an ABox.

- *Executability:* $S_1, \ldots, S_k$ is *executable in $\mathcal{A}$ w.r.t. $\mathcal{T}$* iff the following condition is true for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$: (1) $\mathcal{I} \models \mathsf{pre}_1$ and (2) for all i with $1 \leq i < k$ and all interpretations $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{S_1,\ldots,S_i}^{\mathcal{T}} \mathcal{I}'$, we have $\mathcal{I}' \models \mathsf{pre}_{i+1}$.
- *Projection:* an assertion φ is a *consequence of applying $S_1, \ldots, S_k$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$* iff, for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$, and all $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{S_1,\ldots,S_k}^{\mathcal{T}} \mathcal{I}'$, we have $\mathcal{I}' \models \varphi$.

△

Note that executability alone does not guarantee that we cannot get stuck while executing a composite service. This cannot happen if we additionally know that all services S_i are *consistent with $\mathcal{T}$* in the following sense: S_i is not inconsistent with any model $\mathcal{I}$ of $\mathcal{T}$.

The main aim of this work is to show how the two reasoning tasks executability and projection can be computed, and how their complexity depends on the DL used within our framework.

5.4 Deciding Executability and Projection

We develop reasoning procedures for the reasoning services introduced in and analyze the computational complexity of executability and projection of different fragments of $\mathcal{ALCQIO}$.

It has been shown in [9, 11] that executability and projection can be reduced in PTime to each other. Thus we can restrict the attention to the projection problem. We solve this problem by an approach that is similar to the regression operation used in the situation calculus approach [78]: the main idea is to reduce projection, which considers sequences of interpretations $\mathcal{I}_0, \ldots, \mathcal{I}_k$ obtained by service application, to standard reasoning tasks for single interpretations $\mathcal{I}$.

The theory we obtain can again be expressed by a DL TBox and ABox. This way, projection is reduced to the inconsistency of DL ABoxes, from which we obtain decidability results and upper complexity bounds. Interestingly, when taking this approach, we cannot always stay within the DL we started with since we need to introduce nominals in the reduction.

The following results are proved in [9, 11]:

Theorem 33. *Executability and projection of composite services w.r.t. acyclic TBoxes are*

1. PSpace*-complete for $\mathcal{ALC}$, $\mathcal{ALCO}$, $\mathcal{ALCQ}$, and $\mathcal{ALCQO}$ if numbers in number restrictions are coded in unary;*
2. ExpTime*-complete for $\mathcal{ALCI}$ and $\mathcal{ALCIO}$;*
3. *co-*NExpTime*-complete for $\mathcal{ALCQI}$ and $\mathcal{ALCQIO}$, regardless of whether numbers in number restrictions are coded in unary or binary.*

Thus, in all cases considered, the complexity of executability and projection for a DL $\mathcal{L}$ coincides with the complexity of inconsistency of ABoxes in $\mathcal{LO}$, the extension of $\mathcal{L}$ with nominals.

Reduction to DL Reasoning

Projection in fragments $\mathcal{L}$ of $\mathcal{ALCQIO}$ can be reduced to ABox (in)consistency in the extension $\mathcal{LO}$ of $\mathcal{L}$ with nominals as it was shown in [9]. We assume unary coding of numbers in number restrictions.

Theorem 34. *Let $\mathcal{L} \in \{\mathcal{ALC}, \mathcal{ALCI}, \mathcal{ALCO}, \mathcal{ALCIO}, \mathcal{ALCQ}, \mathcal{ALCQO}, \mathcal{ALCQI}, \mathcal{ALCQIO}\}$. Then projection of composite services formulated in $\mathcal{L}$ can be polynomially reduced to ABox consequence in $\mathcal{LO}$ w.r.t. acyclic TBoxes.*

The main idea of the reduction is to define $\mathcal{A}_{\mathsf{red}}$ and $\mathcal{T}_{\mathsf{red}}$ such that each *single* model of them encodes a *sequence* of interpretations $\mathcal{I}_0, \dots, \mathcal{I}_n$ obtained by applying $S_1, \dots, S_n$ in $\mathcal{A}$ (and *all* such sequences are encoded by reduction models). Since the size of $\mathcal{A}_{\mathsf{red}}$, $\mathcal{T}_{\mathsf{red}}$, and φ_{red} are clearly polynomial in the size of the input, for the DLs $\mathcal{L}$ considered in Theorem 34, upper complexity bounds for ABox consequence in $\mathcal{LO}$ carry over to projection in $\mathcal{L}$. Many such upper bounds are available from the DL literature.

Lower complexity bounds carry over from ABox consequence in a DL $\mathcal{L}$ to projection in the same DL: $\mathcal{A}, \mathcal{T} \models \varphi$ iff φ is a consequence of applying the empty service $(\emptyset, \emptyset, \emptyset)$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$. Thus, we obtain tight bounds for projection in those DLs $\mathcal{L}$ where the addition of nominals does *not* increase the complexity of reasoning.

Corollary 35. *Executability and projection w.r.t. acyclic TBoxes are*

1. PSPACE-*complete for $\mathcal{ALC}$, $\mathcal{ALCO}$, $\mathcal{ALCQ}$, and $\mathcal{ALCQO}$;*
2. *in* EXPTIME *for $\mathcal{ALCI}$;*
3. EXPTIME-*complete for $\mathcal{ALCIO}$;*
4. *in co-*NEXPTIME *for $\mathcal{ALCQI}$;*
5. *co-*NEXPTIME-*complete for $\mathcal{ALCQIO}$.*

Alternatively to the reduction to ABox reasoning, a reduction to reasoning in C2 can be used to obtain these results, see [9].

Hardness Results

We have to consider cases where ABox inconsistency in $\mathcal{LO}$ is harder than in $\mathcal{L}$: we prove an EXPTIME lower bound for projection in $\mathcal{ALCI}$ and a co-NExpTime lower bound for projection in $\mathcal{ALCQI}$ with numbers coded in unary. These bounds carry over to executability. The results show that the additional complexity that is obtained by introducing nominals in the reduction of projection to ABox consequence cannot be avoided.

The idea for proving the lower bounds is to reduce, for $\mathcal{L} \in \{\mathcal{ALCI}, \mathcal{ALCQI}\}$, unsatisfiability of $\mathcal{LO}$ concepts to projection in $\mathcal{L}$. In the case of $\mathcal{ALCQI}$, we can even obtain a slightly stronger result by reducing concept unsatisfiability in $\mathcal{ALCFIO}$ to projection in $\mathcal{ALCFI}$, where $\mathcal{ALCFIO}$ is $\mathcal{ALCQIO}$ with numbers occurring in number restrictions limited to $\{0, 1\}$, and $\mathcal{ALCFI}$ is obtained from $\mathcal{ALCFIO}$ by dropping nominals.

Theorem 36. *There exists an ABox $\mathcal{A}$ and an atomic service S formulated in $\mathcal{ALCI}$ ($\mathcal{ALCFI}$) such that the following tasks are* EXPTIME-*hard (co-*NEXPTIME-*hard): given an ABox assertion φ,*

- *decide whether φ is a consequence of applying S in $\mathcal{A}$;*
- *decide whether $S, (\{\varphi\}, \emptyset, \emptyset)$ is executable in $\mathcal{A}$.*

Note that we cannot obtain the same hardness results for executability of *atomic* services: (i) executability of atomic services in any DL $\mathcal{L}$ can be trivially reduced to ABox (in)consistency in $\mathcal{L}$, and (ii) the complexity of ABox consistency is identical to the complexity of concept satisfiability in $\mathcal{ALCI}$ and $\mathcal{ALCFI}$.

Problematic Extensions

In the DL framework for reasoning about services proposed in this paper, we have adopted several syntactic restrictions: (1) we do not allow for transitive roles; (2) we only allow for acyclic TBoxes and (3) in post-conditions $\varphi/C(a)$, we require C to be a primitive concept or its negation.

Transitive roles introduce a kind of non-determinism, e.g. for services with an empty set of occlusions, that non-determinism of this kind requires extra effort to obtain sensible consequences of action/service executions. Thus, we need a mechanism for eliminating unwanted outcomes or preferring the desired ones.

Cyclic TBoxes and GCIs admitted in the TBoxes give rise to semantic problems: for acyclic TBoxes, the interpretation of primitive concepts *uniquely* determines the extension of the defined ones, while this is not the case for cyclic ones. Together with the fact that the preference relation between interpretations $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$ only takes into account primitive concepts, this means that the minimization of changes induced by service application does not work as expected.

However, the problems are even more serious in the case of GCIs: first, GCIs do not allow an obvious partitioning of concept names into primitive and defined ones. Thus, in the definition of $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$, the only choice is to minimize *all* concept names, which corresponds to the problematic minimization of complex concepts mentioned above. Second, the missing distinction between primitive and defined concepts means that we can no longer restrict concepts C in post-conditions $\varphi/C(a)$ to literals over *primitive* concept names. The best we can do is to restrict such concepts to literals over arbitrary concept names.

Thus, it seems that GCIs cannot be admitted without simultaneously admitting arbitrarily complex concepts in post-conditions.

Complex concepts in Post-Conditions Let a *generalized* service be a service where its post-condition ψ is no longer restricted to be a literal over primitive concepts.

As discussed in [9], there are both semantic and computational problems with generalized services: firstly, they offer an expressivity that is difficult to control and often yields unexpected consequences. Secondly, reasoning with generalized services easily becomes undecidable.

5.5 Related Work

Besides the above framework for matching of Web services and modeling and reasoning for actions in DLs, there has been some work on related tasks. Most of it was done for planning applications, where different reasoning services were considered. Another line of research is dedicated to the process of and the reasoning tasks for Matchmaking of the service descriptions.

One of the early approaches to combine DLs and actions was presented in [88]. The authors want to solve the task of plan recognition and propose means to compute subsumption between plans. In their approach plans consist of complex constraint networks, which are composed of steps, i.e., complex action concepts and sets of Allen relations to specify the temporal relations between these steps. Their plan subsumption algorithm is a hybrid method of computing a mapping between the graph structures of the constraint networks and terminological reasoning w.r.t. the complex concepts in the nodes of the constraint networks.

A quite similar formal framework that permits dealing with actions and plans in a more uniform way was proposed in [3] and elaborated in [4]. Here actions are temporal constraints on world states, which describe how the world is affected by the occurrence of actions. Plans, in turn, are complex actions described as a collection of action types constrained by interval-based temporal relations. In contrast to the approach presented in [88], here temporal relations and temporal variables are directly part of the concept language and ABox statements. Thus the classical DL reasoning services as subsumption and instance checking suffice to realize plan description classification and specific plan recognition. However, the concept definitions in the terminologies supported by this approach have to be acyclic and unique. Plan subsumption can be reduced to concept subsumption between non-temporal concepts and to subsumption between temporal constraint networks. The authors showed that classical DL reasoning for their interval-based temporal DLs are decidable [4]. Furthermore they showed that for their temporal extension of $\mathcal{ALC}$ subsumption is NP-complete.

The approach proposed in [62] for inferring subsumption between actions is based on a different way of characterizing actions in DLs. Here the main ingredients are action concepts in the terminology that are associated with pre- and post-conditions and that do not have an explicit time representation. These conditions are expressed in a subset of the concept language. To infer subsumption relation between action concepts they propose a set of rules between the pre- and post-conditions.

A similar way of dealing with subsumption of action concepts is described in [56]. Here the representation of action concepts is also based on pre- and post-conditions, expressed in the concept language. More precisely, the pre- (and post-)conditions of an action are sets of world states that are required (caused) by this action modeled by dynamic roles and features. The subsumption of two action concepts is then defined as the inclusion of pre- and post-conditions respectively.

Besides the approaches that focus on the description of actions, there are also approaches dedicated to the task of matchmaking of services. The service descriptions are mostly simple concept descriptions in DLs that do not provide special constructs for expressing services. The reasoning algorithms are often a clever application of DL standard reasoning services.

In [61] a framework proposed by [85] has been elaborated. In both works the tasks of service advertisements, service discovery and matchmaking are addressed. The advertised services as well as the service requests are written in an ontology language (DAML-S or

DAML+Oil resp.) and matching combinations of requests and advertisements are to be computed automatically. To this end it is proposed in [61] to describe services by service profiles, that capture (among others) information on pre-conditions, effects and outputs of the service. The matchmaking for a request R, i.e., the finding of the advertisements that potentially satisfy the requirements specified in R, is realized by finding equivalent, more general or more specific service advertisement concepts than R. Furthermore Disjointness (Intersection) of concepts is used to detect incompatible (compatible) combinations of requests and advertisements.

In [37] this framework was again extended. The authors concentrate on handling variance inherent to service descriptions. Here a service description is a *set of* concept axioms, where one axiom from the set defines the service concept S, which enables more precise characterizations of services and thus probably better matching results.

5.6 Discussion

Standard problems in reasoning about action (projection, executability) become decidable if one restricts the logic for describing pre- and post-conditions as well as the state of the world to certain decidable DLs $\mathcal{L}$. The complexity of these inferences is determined by the complexity of standard DL reasoning in $\mathcal{L}$ extended by nominals.

This is only a first proposal for a formalism describing the functionality of Web services, which must be extended in several directions. First, instead of using an approach similar to regression to decide the projection problem, one could also try to apply *progression*, i.e., to calculate a successor ABox that has, as its models, all the successors of the models of the original ABox. Second, the expressiveness of the basic action formalism should extended. Either one can explore how the restriction w.r.t to complex post-conditions and general TBoxes can be relaxed, while still staying decidable. Furthermore the basic action formalism introduced by Reiter has been extended in several directions, and we need to check for which of these extensions our results still hold. Third, we have used only composition to construct composite services, whereas OWL-S proposes also more complex operators. These could, for example, be modeled by appropriate GOLOG programs. Finally, to allow for automatic composition of services, one would need to look at how planning can be done in our formalism.

6 Configuration of Technical Devices

As one of the first commercial applications of ontologies, configuration systems for technical devices have been designed and implemented (see, e.g., [68, 67]). The key idea is to use an ontology (TBox) to describe the configuration space and an initial configuration (ABox) to be automatically completed such that given constraints are met. The goal is to derive a formalization of ontology-based configuration as a formal decision problem in order to avoid the error-prone development of special-purpose programs. However, early approaches were based on inexpressive ontology languages, i.e., many constraints could not be expressed in a declarative way, and a lot of programming was still required.

With expressive ontology languages, domain-specific constraints can be declaratively expressed and with expressive query languages, many properties of the final configuration can be queried, maybe resulting in further constraints being declared. The key benefit is that a declarative specification of the configuration space provides for enhanced flexibility if requirements change. Descriptions of the configuration space can be evaluated during the design phase, and domain-specific vocabulary can be captured by the ontology.

6.1 Formalizations of the Configuration Task

Early approaches to formalize the configuration task with ontologies used description logic ontologies and treated configuration as a consistency maintenance task with an expressive representation language but incomplete DL reasoner [76]. Later, an approach with a complete reasoner for a rather inexpressive language was pursued [68]. As description logic technology matured, industrial applications were in reach. Since the reasoner was only used for checking consistency (or satisfiability) of an ABox representing the constructed artifact, most of the program code for exploring the configuration space had to be manually written. Nevertheless, the declarative ontology was used to guide the search. However, much of configuration problem solving remained outside of the logic.

In another approach configuration was formalized using a logical decision problem, namely satisfiability checking in general, and model generation in particular. The idea was to define the final configuration as a model of a description of the initial configuration (ABox) w.r.t. to a TBox [16, 17]. In this approach it is particularly important to avoid unintended models of the knowledge base, and hence, an expressive languages is required (see [5] for cardinality restrictions on concepts or [89] for a closing operator for taxonomies).

A concrete system implementation that does not only prove that a model exists, but can actually generate one or even multiple models is still not available. Acknowledging the lack of this technology, a planning-based approach to configuration with description logic was suggested in [26, 27]. Rather than using programs to explore the configuration space, in this work a planning system is employed to enumerate possible configurations which, then, are checked for satisfiability with a description logics system (Racer, c.f., [26, 27]).

Nevertheless, model generation for description logic knowledge bases would still have important applications in industrial contexts, not only for configuration but also for other tasks (see, e.g., [81]).

6.2 Challenges for Ontology Processing

For computing models of an ontology based on an expressive language, there are several challenges to face:

1. Models can be infinite (e.g., this holds for $\mathcal{SHIQ}$). Thus, what can be compute is only a model description, not directly a model.

2. Models can be very large. In a server-based architecture, retrieving a complete model can be very time-consuming.

3. Tableau-based prover systems generate models, but only canonical models. Canonical models are least-commitment models and might not necessarily correspond to the models that users would like to have computed. It is also not trivial to enumerate all models.

In order to solve these problems, new research is required (e.g., to define and implement a model query and manipulation language).

References

[1] Serge Abiteboul. Towards a Deductive Object-Oriented Database Language. *Data and Knowledge Engineering*, 5:263–287, 1990.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[3] Alessandro Artale and Enrico Franconi. A Computational Account for a Description Logic of Time and Action. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 3–14, 1994.

[4] Alessandro Artale and Enrico Franconi. A Temporal Description Logic for Reasoning about Actions and Plans. *J. of Artificial Intelligence Research*, 9:463–506, 1998.

[5] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality Restrictions on Concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[6] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[7] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[8] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics as Ontology Languages for the Semantic Web. In Dieter Hutter and Werner Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

[9] Franz Baader, Carsten Lutz, Maja Milicic, Ulrike Sattler, and Frank Wolter. A Description Logic Based Approach to Reasoning about Web Services. In *Proc. of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.

[10] Franz Baader, Carsten Lutz, Maja Milicic, Ulrike Sattler, and Frank Wolter. Integrating Description Logics and Action Formalisms: First Results. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, Pittsburgh, PA, USA, 2005.

[11] Franz Baader, Maja Milicic, Carsten Lutz, Ulrike Sattler, and Frank Wolter. Integrating Description Logics and Action Formalisms for Reasoning about Web Services. LTCS-Report LTCS-05-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[12] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.

[13] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML Class Diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[14] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A Structural Data Model for Objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 59–67, 1989.

[15] Kerstin Bücher, Yves Forkl, Günther Görz, Martin Klarner, and Bernd Ludwig. Discourse and Application Modeling for Dialogue Systems. In *In Proc. 2001 Workshop on Applications of Description Logics*, 2001. `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-44/`.

[16] Martin Buchheit, Rudiger Klein, and Werner Nutt. Configuration as Model Construction: the Constructive Problem Solving Approach. In *Proc. of the 4th Int. Conf. on Artificial Intelligence in Design*, Lausanne (Switzerland), August 1994.

[17] Martin Buchheit, Rüdiger Klein, and Werner Nutt. Constructive Problem Solving: A Model Construction Approach towards Configuration. Technical Report DFKI Technical Memo TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, 1995.

[18] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data Complexity of Query Answering in Description Logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.

[19] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[20] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering Queries Using Views over Description Logics Knowledge Bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[21] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information Integration: Conceptual Modeling and Reasoning Support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

[22] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[23] Tiziana Catarci, Paolo Dongilli, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, and Sergio Tessaris. An Ontology based Visual Tool for Query Formulation Support. In *Proc. of the 16th eur. conf. on artificial intelligence (ecai 2004)*, 2004.

[24] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, Berlin (Germany), 1990.

[25] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in Concept Languages: From Subsumption to Instance Checking. *J. of Logic and Computation*, 4(4):423–452, 1994.

[26] Michael Eisfeld. Model Construction for Configuration Design. In *Proceedings of the Workshop of Applications of Description Logics, German Conference on Artificial Intelligence*, 2002. `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-63/`.

[27] Michael Eisfeld and Raimar Scherer. Assisting Conceptual Design of Building Structures by an Interactive Description Logic-based Planner. *Advanced Engineering Informatics*, 17(1):41–57, 2003.

[28] Richard Fikes, Patrick Hayes, and Ian Horrocks. OWL-QL: A Language for Deductive Query Answering on the Semantic Web. *J. of Web Semantics*, 2(1), 2005.

[29] Enrico Franconi, Sergio Tessaris, Tiziana Catarci, Tania Di Mascio, Giuseppe Santucci, and Guido Vetere. Specification of the Ontology Design Tool. Technical Report D6.2, SEWASIE Consortium, 2003.

[30] Malte Gabsdil, Alexander Koller, and Kristina Striegnitz. Building a Text Adventure on Description Logic. In *In Proc. 2001 Workshop on Applications of Description Logics*, 2001. `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-44/`.

[31] Jan Galinski, Atila Kaya, and R. Möller. Development of a Server to Support the Formal Semantic Web Query Language OWL-QL. In *Proc. of the Int. Workshop on Description Logics, DL '05*, 2004.

[32] Michael R. Garey and David S. Johnson. *Computers and Intractability — A guide to NP-completeness*. W. H. Freeman and Company, San Francisco (CA, USA), 1979.

[33] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On the Update of Description Logic Ontologies at the Instance Level. In *Proceedings of the 21th National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, AAAI'06*. AAAI Press, 2006.

[34] Matthew L. Ginsberg and David E. Smith. Reasoning About Action I: A Possible Worlds Approach. *Artif. Intell.*, 35(2):165–195, 1988.

[35] Birte Glimm and Ian Horrocks. Query Answering Systems in the Semantic Web. In *Proc. of the KI-04 Workshop on Applications of Description Logics 2004, ADL '04*, 2004.

[36] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Conjunctive Query Answering for Description Logics with Transitive Roles. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[37] Stephan Grimm, Boris Motik, and Chris Preist. Variance in E-Business Service Discovery. In *Proceedings of the ISWC 2004 Workshop on Semantic Web Services*, 2004.

[38] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, pages 48–57, 2003.

[39] Volker Haarslev and Ralf Möller. RACER System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001. Available at `http://www.racer-systems.com`.

[40] Volker Haarslev and Ralf Möller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, pages 85–94, 2003.

[41] Volker Haarslev and Ralf Möller. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004.

[42] Volker Haarslev, Ralf Möller, Ragnhild Van Der Straeten, and Michael Wessel. Extended Query Facilities for Racer and an Application to Software-Engineering Problems. In *Proc. of the Int. Workshop on Description Logics, DL '04*, 2004.

[43] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the Semantic Web with Racer + nRQL. In *Proc. of the KI-04 Workshop on Applications of Description Logics 2004, ADL '04*, 2004.

[44] Peter Haase and Ljiljana Stojanovic. Consistent Evolution of OWL Ontologies. In Asuncin Gmez-Prez and Jrme Euzenat, editors, *Proceedings of the Second European Semantic Web Conference, Heraklion, Greece, 2005*, volume 3532 of *Lecture Notes in Computer Science*, pages 182–197. Springer, may 2005.

[45] Christian Halaschek-Wiener, Bijan Parsia, and Evren Sirin. Description Logic Reasoning with Syntactic Updates. In *Proceedings of the 5th Int. Conference on Ontologies, Databases, and Applications of Semantics, ODBase'06*, 2006. Submitted.

[46] Christian Halaschek-Wiener, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Description Logic Reasoning for Dynamic ABoxes. In *Proceedings of the Int. Workshop on Description Logics, DL'06*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[47] Jeff Heflin and James Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[48] Bernhard Hollunder. Consistency Checking Reduced to Satisfiability of Concepts in Terminological Systems. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.

[49] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

[50] Ian Horrocks and Ulrike Sattler. Optimised Reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281, July 2002.

[51] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to Decide Query Containment Under Constraints Using a Description Logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[52] Ian Horrocks and Sergio Tessaris. A Conjunctive Query Language for Description Logic ABoxes. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[53] Ian Horrocks and Sergio Tessaris. Querying the Semantic Web: a Formal Approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[54] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing $\mathcal{SHIQ}$-Description Logic to Disjunctive Datalog Programs. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, 2004.

[55] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[56] Christel Kemke. A Formal Theory for Describing Action Concepts in Terminological Knowledge Bases. In *Canadian Conference on AI (AI 2003)*, volume 2671 of *LNCS*, pages 458–465. Springer, 2003.

[57] Jinyoul Lee, Keng Siau, and Soongoo Hong. Enterprise Integration with ERP and EAI. *Communications of the ACM*, 46(2):54–60, 2003.

[58] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[59] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Programming*, 31:59–84, 1997.

[60] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[61] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. *Int. J. of Electronic Commerce*, 8(4):39–60, 2004.

[62] Thorsten Liebig and Dietmar Rösner. Action Hierarchies in Description Logics. In *Proc. of the 1997 Description Logic Workshop (DL'97)*. AAAI Press, 1997.

[63] Thorsten Liebig, Anni-Yasmin Turhan, Olaf Noppens, and Timo Weithöner. DIG 2.0 Proposal for Accessing Told Data. Available as `http://www.informatik.uni-ulm.de/ki/Liebig/told-access.html`, May 2006.

[64] Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating Description Logic ABoxes. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings of the Tenth Int. Conference on Principles of Knowledge Representation and Reasoning, KR'06*, pages 46–56. AAAI Press, 2006.

[65] Robert MacGregor. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, 1991.

[66] Robert M. MacGregor and David Brill. Recognition Algorithms for the Loom Classifier. In *Proc. of AAAI'92, Thenth Conference on Artificial Intelligence*, 1992.

[67] Deborah L. McGuinness. Configuration. In Baader et al. [6], chapter 12, pages 388–405.

[68] Deborah L. McGuinness and Jon R. Wright. An industrial-strength Description Logic-based Configurator Platform. *IEEE Intelligent Systems*, 13(4):69–77, 1998.

[69] Sheila McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems. Special Issue on the Semantic Web*, 16(2):46–53, 2001.

[70] Ralf Möller, Volker Haarslev, and Michael Wessel. On the Scalability of Description Logic Instance Retrieval. In Chr. Freksa and M. Kohlhase, editors, *29. Deutsche Jahrestagung für Künstliche Intelligenz*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2006.

[71] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. In *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, November 2004.

[72] Bernd Neumann and Ralf Möller. On Scene Interpretation with Description Logics. In H.I. Christensen and H.-H. Nagel, editors, *Cognitive Vision Systems: Sampling the Spectrum of Approaches*, number 3948 in LNCS, pages 247–278. Springer, 2006.

[73] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Characterizing Data Complexity for Conjunctive Query Answering in Expressive Description Logics. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, 2006.

[74] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data Complexity of Answering Unions of Conjunctive Queries in $\mathcal{SHIQ}$. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[75] Maria Magdalena Ortiz de la Fuente, Diego Calvanese, Thomas Eiter, and Enrico Franconi. Data complexity of answering conjunctive queries over $\mathcal{SHIQ}$ knowledge bases. Technical report, Faculty of Computer Science, Free University of Bozen-Bolzano, July 2005. Also available as CORR technical report at `http://arxiv.org/abs/cs.LO/0507059/`.

[76] Bernd Owsnicki-Klewe. Configuration as a Consistency Maintenance Task. In *Proc. of the 12th German Workshop on Artificial Intelligence (GWAI'88)*, pages 77–87. Springer, 1988.

[77] Bijan Parsia, Christian Halaschek-Wiener, and Evren Sirin. Towards Incremental Reasoning Through Updates in OWL-DL. Available as `http://www.mindswap.org/papers/2006/incclass.pdf`, 2006.

[78] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[79] Mathieu Roger, Ana Simonet, and Michel Simonet. Toward Updates in Description Logics. In *Proceedings of the Int. Workshop on Description Logics, DL'02*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2002.

[80] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[81] Carsten Schröder and Bernd Neumann. On the Logics of Image Interpretation: Model Construction in a Formal Knowledge-Representation Framework. *IEEE International Conference on Image Processing. IEEE Computer Society Press*, 1996.

[82] Murray Shanahan. Perception as Abduction: Turning Sensor Data into Meaningful Representation. *Cognitive Science*, 29:103–134, 2005.

[83] Evren Sirin and Bijan Parsia. Pellet System Description. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2006.

[84] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[85] David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A Semantic Web Approach to Service Description for Matchmaking of Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001.

[86] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, 2006. To appear.

[87] Moshe Y. Vardi. The Complexity of Relational Query Languages. In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[88] Robert Weida and Diane Litman. Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 282–293, 1992.

[89] Robert A. Weida. Closed Terminologies in Description Logics. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 592–599, 1996.

[90] Marianne Winslett. Reasoning About Action using a Possible Models Approach. In *AAAI*, pages 89–93, Saint Paul, MN, 1988.