Techniques for Ontology Access, Processing, and Usage

Deliverable TONES-D18

Diego Calvanese[1], Giuseppe De Giacomo[2], Birte Glimm[3], Bernardo Cuenca Grau[3],
Volker Haarslev[5], Ian Horrocks[3], Alissa Kaplunova[5],
Domenico Lembo[2], Maurizio Lenzerini[2], Carsten Lutz[4],
Maja Milicic[4], Ralf Möller[5], Riccardo Rosati[2],
Ulrike Sattler[3], Michael Wessel[5]

[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:	D18.pdf
Version:	1.1
Date:	July, 2007
Number of pages:	143

Abstract

This report summarizes the reasoning techniques and algorithms developed in the TONES project that realize the reasoning services identified as fundamental for ontology-based access, processing, and usage. In addition to a detailed presentation of the algorithms, we report on their computational properties and investigate solutions for the expressivity and data scalability problems.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	2007	First draft
v.1.1	2007	Final version

Contents

1 Introduction

In this deliverable, we describe techniques and algorithms used for performing tasks for ontology-based access, processing, and usage, that have been identified in the previous deliverable (see [29]). We start with an overview on the mentioned tasks and present in the next sections our main contributions for this working package.

1.1 Tasks

Query Answering 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. We focus on query languages in the context of ontologies based on description logics. Query languages based on so-called grounded conjunctive queries restrict the variables in queries to be bound to individual names in the ABox only. Query languages supporting unrestricted conjunctive queries for very expressive ontology languages have been investigated recently.

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. For information access we distinguish two different scenarios: i) 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, focus is on inferred information (the standard view, no distinction between explicitly given information and derived information) or focus is on explicitly stated information only (aka told information access); ii) information is implicit in media data, and logical descriptions of the content must be derived in beforehand.

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 is supported by automated reasoning tasks which make use of the ontologies describing the data sources.

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. 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.

Semantic Service Discovery and Selection In addition to static information, the Web also offers services which allow their users to effect changes in the world. 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 addition to atomic services, we also consider simple composite services, which are sequences of atomic services.

Configuration of Technical Devices 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 deci-

sion 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.

Ontology-based information systems provide a framework for studying representation languages as well as query languages from a theoretical and a practical point of view. Many of the above-mentioned tasks for ontology access, processing, and usage must be realized in ontology-based information systems. In this deliverable the techniques for tasks query answering and semantic service discovery and selection are discussed in detail. Techniques used for solving tasks of information extraction and configuration of technical devices are described e.g., in [91] and [53].

1.2 Ontology-based Information Systems (OBISs)

It is now widely accepted that ontologies will play an important role for the next generation of information systems (ISs). The use of ontologies for ISs will not only enable "better" and "smarter" retrieval facilities than current ISs based on the predominant relational data model (cf. the vision of the Semantic Web [21]), but also play a key role in supporting data and information quality checks, IS interoperability, and information integration [41, 42, 22]. Ontologies provide the means for solving problems raised by semantic heterogeneity in ISs based on different conceptual or logical data models, because ontologies inherently work on a semantic rather than on a syntactic level and thus support a seamless incorporation of conceptual domain constraints into the machinery of an information system [151].

Mostly for performance reasons, retrieval systems nowadays still use rather simple thesaurus-based retrieval models (possibly based on statistical information) [50]. From a logical point of view, a thesaurus-based system uses a rather inexpressive representation formalism. Recent developments in Description Logic inference technology have shown that expressive formalisms can indeed be used for building practical systems in general, and practical information systems in particular. For instance, information retrieval systems based on Description Logics are described in [80, 110]. We broaden this view and describe in the next sections a formal generic framework for building ontology-based information systems (OBISs). As such, our framework must offer the means for (i) the extensional layer, (ii) the intensional layer, and (iii) the query component. Being ontology-based, our framework is strongly influenced by Description Logics (DLs) and offers novel solutions for certain problems we have encountered during our endeavor of implementing OBISs with a standard DL system.

We make these problems transparent by means of a case study: design and implementation of an ontology-based geographic information system (GIS). Based on our framework we present empirically successful solutions for problems in this specific IS domain. The focus in the case study is on ontology-based query answering. The DLMAPS system supports ontology-based spatio-thematic query answering for city maps [157, 156].

Up to now the number of implemented OBISs is rather small, however. Consequently, experience with the scalability of the DL approach is limited. This is not surprising, since DL systems are a rather new technology compared to databases and, as we will see in the following, some problems remain to be solved in today's DL technology.

1.2.1 Problem Identification

We have identified 7 main problems **P1 – P7** which contribute to the difficulties we encountered regarding the use of DL systems for building OBISs (also see [32]). The problems **P1, P2** are DL-specific, whereas **P3 – P6** are specific to the APIs of contemporary DL systems. **P7** concerns the software architecture of DL systems.

We believe that DLs have their deficiencies regarding expressivity and are not a panacea for arbitrary

information modeling and representation (of course, this holds for all formalisms). DLs are very well suited for the representation of semi-structured (or even incomplete/uncertain) information [38], but things become more complicated if special "non-abstract" domains such as space are considered (→ **P1:** DL applicability problem). We say "non-abstract" since space has a rich inner natural structure which is not "man-made", but is given by the laws of physics. Here, either non-standard DLs or non-trivial logical encodings are needed. For these non-standard DLs, no working systems exist, and in our experience, complex logical encodings are very likely to decrease the performance of the reasoning component.

Due to the well-known *expressivity vs. complexity tradeoff,* sound and complete reasoning with expressive DLs is not a trivial task. *Data scalability* is not always easy to achieve for expressive DLs. There exist inexpressive ontology languages such as, for instance, RDF(S) [106, 26, 34] which scale well regarding data complexity. However, these approaches fail to scale regarding expressivity and problems then have to be solved outside the information system by resorting to programming. We believe that a generic framework for building OBISs should be parameterizable in both dimensions: data scalability and expressivity scalability (→ **P2**: data and expressivity scalability problem) [63, 112]. If high expressivity is required, it should be supported. However, if only low expressivity is needed, then the user should not have to pay the higher price if a reasoner was used which implements a much more expressive logic (see, e.g., the case study in [90]). This implies that a reasoner should be selected that implements *just the required logic*, so that the lower complexity bound is sufficient, and the upper bound is tight.

DL systems somehow live in their own realm and are thus not really interoperable with the rest of the more conventional IS infrastructure, e.g., existing relational database technology (→ **P3:** interoperability and middleware problem). However, due to the inherent intellectual complexity of building DL system, existing DL systems must be reused and exploited as *componentware* if possible.

Even though standards such as DIG exist [19], it can be observed that for building practical OBIS some API functionality is still missing, only part of which is currently about to be standardized in $\text{DIG}_{2.0}$ [145]. Compared with the APIs found in *relational database management systems (RDMSs)*, one can observe that functionality regarding the management of the physical schema or storage layer of a DL system is missing (→ **P4:** missing storage-layer-functionality problem).

Moreover, as for RDMSs, *plug-in mechanisms or "stored procedures"* would be beneficial in order to open up the server architectures for applications as well as to achieve high-bandwidth communication. *Extensibility and openness* is not yet achieved in standard DL systems [157, 145] (→ **P5:** extensibility and openness problem). Even though there is an extension proposal for $\text{DIG}_{2.0}$, which we believe is a very promising idea, $\text{DIG}_{2.0}$ still does not support functionality or API functions to be added to a DL system by users (i.e., application builders). It is clear that this problem can only be addressed by some kind of programming facility or plug-in mechanism.

Only recently, expressive *query languages (QLs)* have been investigated and incorporated into DL systems (→ **P6:** missing QL problem) [47, 97, 84, 156, 56]. However, these are indispensable for OBIS.

The last problem (**P7**) is closely related to **P2** and concerns the *software architecture* of a reasoner. Although reasoners implementing highly expressive logics are also capable to processing KBs utilizing only a (less expressive) sub-logic, one can sometimes observe that specialized reasoners crafted to support smaller logics perform better than reasoners supporting more expressive logics. From the perspective of the more expressive reasoner, the more efficient (and more specialized) inference algorithm implemented by the less expressive reasoner can be seen as an optimization technique. In principle, the performance of the more expressive reasoner can become comparable once a specialized optimization is built in. With more and more dedicated optimizations, whose applicability must be automatically detected, however, the maintenance of the DL system software becomes a serious problem. We believe that it is important to have *appropriate software abstractions* which help to maintain the software and manage the complexity introduced by language-specific optimization techniques.

Specialized reasoning algorithms are not only needed in order to realize special optimizations, but

also to implement certain inference tasks. *From a theoretical perspective,* most expressive DL systems "only" have to implement a reasoner to decide one core inference problem, e.g., an ABox satisfiability checker since the other inference problems are *reducible* to the core problem. However, *from a computational perspective*, this seems inadequate because highly dedicated algorithms for special inference problems have to be used to ensure scalability, e.g., for the *instance retrieval problem* [61]. These algorithms are sometimes even more complex than tableau calculi [17], and thus *deserve a clean separation* from other parts of the system code in order to achieve maintainability. Again, appropriate domain-specific software abstractions are needed. We thus call **P7** the "software-abstraction problem".

1.2.2 Layered vs. Integrated Approaches

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research. Although description logics become more and more expressive (e.g., [72]), it is also necessary to be able to deal with huge amounts of data descriptions very effectively. Thus, in practice, description logic systems offering high expressivity must also be able to handle large bulks of data descriptions derived from database content. Users expect that DL systems scale w.r.t. these practical needs. We consider two kinds of scalability problems: scalability w.r.t. large sets of data descriptions (data description scalability), i.e., runtimes scale well with increased data sizes but unchanged conceptual descriptions, and scalability w.r.t. increased expressivity (expressivity scalability), i.e., a reasoner can still process an ontology in reasonable time if the data size remains unchanged but more complex conceptual descriptions (e.g., full negation, disjunction) were added.

In the literature, the data description scalability problem has been tackled from different perspectives. We see two main approaches, the layered approach and the integrated approach. In the layered approach the goal is to use databases for storing and accessing data. From the point of view of the RDMS, the inference algorithms then have to reside in the *application layer*.[1] Description logic ontologies are then exploit for convenient query formulation. The main idea is to support ontology-based query translation to relational query languages (SQL, datalog). See, e.g., [162, 59] (DLDB), [18] (Instance Store), [152] (deductive databases), or [30] (*DL-Lite*). We notice that these approaches are only applicable if reduced expressivity for conceptual descriptions does not matter. Despite the most appealing argument of reusing database technology (in particular services for persistent data), at the current state of the art it is not clear how expressivity can be increased to, e.g., $\mathcal{SHIQ}$ without losing the applicability of transformation approaches. Hence, while data description scalability is achieved, it is not clear how to extend these approaches to achieve expressivity scalability (at least for some parts of the data descriptions). In fact, since *ontology-based query answering* requires inference, the assertions in the database are used as input assertions for the inference algorithm. Unfortunately, the question *which assertions to retrieve from the database* can only be resolved at runtime by the inference algorithm itself. But if there is no way to tell in advance which assertions will contribute to the final answer of the query and which will not, then database indices are of no great help in order to reduce the set of candidate assertions to consider.

Thus, for expressive ontology languages, a layered architecture results in a lot of communication overhead, and the retrieved candidate results from the database must be combined and reasoned about to get the final query answer. Obviously, it would be better if this computation and integration of required sub-results could be done in the RDMS itself by means of a single query. This is possible as the authors of the QuOnto system have shown, but "only" for rather inexpressive DLs. In the case of QuOnto, ontology-based query answering can be performed by the RDMS query answering engine on its own, since the inexpressivity of the underlying DL makes it possible to expand the original query in such a way that it takes the ontology into account [2].

[1] We think it is unrealistic to assume that a system as complex as a tableaux reasoner can be realized as a stored procedure within a RDMS.

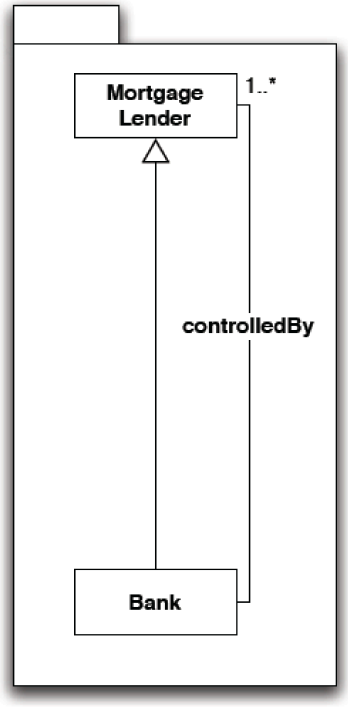

Figure 1: A small example ontology in UML-like notation (the association is described by a cardinality restriction `1..*`).

1.3 Grounded vs. Unrestricted Queries in Description Logics

As we have discussed above, for future ontology-based information systems, query answering is essential. For different practical purposes, different formal semantics of query languages are introduced in the sequel. With respect to ontologies, the class of *conjunctive queries* is investigated, and – as mentioned above – we distinguish between so-called *grounded conjunctive queries* and *unrestricted conjunctive queries*. Grounded queries can be seen as restricted since in grounded queries, variables occurring in queries are bound to named individuals only whereas in unrestricted conjunctive queries, variables not mentioned in the query head are bound to possibly unnamed domain objects. The distinction between both kinds of queries investigated in this report can be explained with an example.

Consider the following ontology specifiied using the description logics. The role $controlledBy$ has $MortgageLender$ as the domain, and $Bank$ as a range. A $Bank$ is a specific $MortgageLender$, and a $MortgageLender$ is controlled by at least one $Bank$. Then, it is explicitly stated that the individual $Halifax$ is controlled by $LeedsBS$. In addition, the individual RBS is an instance of $Bank$.

$$
\begin{array}{lcl}
\exists controlledBy.\top & \sqsubseteq & MortgageLender \\
\top & \sqsubseteq & \forall controlledBy.Bank \\
Bank & \sqsubseteq & MortgageLender \\
MortgageLender & \sqsubseteq & \exists controlledBy.Bank
\end{array}
$$

$$
\begin{array}{l}
controlledBy(Halifax, LeedsBS) \\
Bank(RBS)
\end{array}
$$

A graphical display of the constraints imposed by the Tbox is shown in UML notation in Figure 1. Now, let us consider the query $ans(X) \leftarrow Bank(X), controlledBy(X, Y)$. In this query, all objects are searched that are $Bank$s and are controlled by some object. Due to the domain and range

restrictions of the role $controlledBy$ it follows that $Halifax$ is a $MortgageLender$ and $LeedsBS$ is a $Bank$, respectively. Since $LeedsBS$ is a $Bank$ it is also a $MortgageLender$. Thus, we have three $MortgageLender$s, $Halifax$, $LeedsBS$, and, by definition, RBS. Due to the Tbox, all $MortgageLender$s are controlled by some $Bank$, which is possible not mentioned using a name in the ontology. Thus, for the conjunctive query given above, the result set contains $LeedsBS$, and RBS since $Halifax$ is not necessarily a $Bank$. The example clearly demonstrates that with ontologies, implicit information comes into play.

The fact that RBS is returned as a member of the query result set might be somewhat surprising since the Bank which controls RBS is not explicitly known (i.e., there is no named domain object known). If one expects that variables are bound to named domain objects, grounded conjunctive queries can be used. In this case the result set contains only $LeedsBS$. The ontology is still important since only with the ontology it is possible to derive that $LeedsBS$ is a $Bank$. In this example, this is a very simple inference, though.

The implementation of a query ansering engine for grounded conjunctive queries is "easier" because query answering can be reduced to standard concept-based instance retrieval operations. Unrestricted conjunctive queries are much more complex to handle, in particular for expressive description logics. In this report we start with grounded conjunctive queries and treat unrestricted conjunctive queries afterwards.

2 Grounded Conjunctive Queries: Scalable Instance Retrieval for Expressive DLs

In the introduction, we discussed two possible approaches for building ontology-based information systems, the layered approach and the integrated approach, that address scalability problems in practical OBISs: scalability w.r.t. large data and scalability w.r.t. expressivity. To solve the data description scalability problem, e.g., an RDMS could be used as the storage layer of an OBIS. This would result in a classical layered architecture. However, the drawback of this layered approach is that it is not obvious how to account for expressivity scalability. Therefore, for investigating solutions to both scalability problems, we pursue the integrated approach that considers query answering with a tableau-based description logic system augmented with new techniques inspired from database systems. For the time being we ignore the persistency problem.

The contribution of this section is twofold. By introducing and analyzing practical *instance retrieval algorithms* tested in one of the mature, sound, and complete description logic systems (RACER [60]), which is used in many research projects all over the world, the development of even more powerful semantic web query engines is directly supported. RACER implements the very expressive DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, also known as $\mathcal{SHIQ}(D^-)$ [74, 60], and offers multiple TBoxes, ABoxes as well as expressive concrete domains (of which the OWL "datatypes" are only a subset). Even though it is not clear under which circumstances a reasoning system can be called *empirically successful*, we claim the RACER is such a system given the evidence that it has many academic as well as commercial users.

This section is partially based on [61] but combines and extends the previously reported results with new optimization techniques and new insight derived from other real-world ontologies. On the other hand the contribution also presents and analyzes the main results we have found about how to start solving the scalability problem with tableau-based prover systems given large sets of data descriptions for a large number of individuals and *grounded conjunctive queries*. The optimization techniques might very well be appropriate for general conjunctive queries.

2.1 Preliminaries

2.1.1 The Description Logic $\mathcal{SHIQ}$

For sake of completeness and readability we briefly introduce the description logic $\mathcal{SHIQ}$ (see the tables in Figure 2) using a standard Tarski-style semantics based on an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. We assume a set of concept names C and a set of role names R. The mutually disjoint subsets P and T of R denote non-transitive and transitive roles, respectively ($R = P \cup T$).

If $\mathsf{R}, \mathsf{S} \in R$ are role names, then the terminological axiom $\mathsf{R} \sqsubseteq \mathsf{S}$ is called a *role inclusion axiom*. A *role hierarchy* $\mathcal{R}$ is a finite set of role inclusion axioms. In order to preserve decidability a syntactic restriction holds for the combinability of number restrictions and transitive roles in $\mathcal{SHIQ}$. Number restrictions are only allowed for *simple* roles, i.e., a role is called simple if it has no transitive super-role. The concept name $\top$ ($\bot$) is used as an abbreviation for $\mathsf{C} \sqcup \neg\mathsf{C}$ ($\mathsf{C} \sqcap \neg\mathsf{C}$).

If C and D are concept expressions, then $\mathsf{C} \sqsubseteq \mathsf{D}$ (*generalized concept inclusion* or *GCI*) is a terminological axiom. A finite set of terminological axioms $\mathcal{T}_{\mathcal{R}}$ is called a *terminology* or *Tbox* w.r.t. to a given role hierarchy $\mathcal{R}$.[2] We use $\mathsf{C} \equiv \mathsf{D}$ as an abbreviation for $\{\mathsf{C} \sqsubseteq \mathsf{D}, \mathsf{D} \sqsubseteq \mathsf{C}\}$.

The *concept satisfiability problem* is to decide whether a given concept expression C is *satisfiable* w.r.t. to $\mathcal{T}$ and $\mathcal{R}$, i.e., whether there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}$ satisfies $\mathcal{T}$ and $\mathcal{R}$ and $\mathsf{C}^{\mathcal{I}} \neq \emptyset$.

Let C be a concept expression, R be a role name, O be the set of individual names (disjoint from the set of concepts names and the set of role names), $\mathsf{a}, \mathsf{b} \in O$ be individual names, then $\mathsf{a}:\mathsf{C}$ is called

[2] The reference to $\mathcal{R}$ is omitted in the following.

Syntax	Semantics
Concepts	
A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, A is a concept name
$\neg\mathsf{C}$	$\Delta^{\mathcal{I}} \setminus \mathsf{C}^{\mathcal{I}}$
$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}}$
$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cup \mathsf{D}^{\mathcal{I}}$
$\exists\,\mathsf{R}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \exists\, b \in \Delta^{\mathcal{I}} : (a,b) \in \mathsf{R}^{\mathcal{I}},\ b \in \mathsf{C}^{\mathcal{I}}\}$
$\forall\,\mathsf{R}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \forall\, b : (a,b) \in \mathsf{R}^{\mathcal{I}} \Rightarrow b \in \mathsf{C}^{\mathcal{I}}\}$
$\exists_{\geq n}\,\mathsf{S}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \mathsf{S}^{\sharp}(a,\mathsf{C}) \geq n\}$
$\exists_{\leq m}\,\mathsf{S}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \mathsf{S}^{\sharp}(a,\mathsf{C}) \leq m\}$
Roles	
R	$\mathsf{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
R^{-}	$\{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b,a) \in \mathsf{R}^{\mathcal{I}}\}$

Terminological Axioms	
Syntax	Satisfied if
$\mathsf{R} \in T$	$\mathsf{R}^{\mathcal{I}} = (\mathsf{R}^{\mathcal{I}})^{+}$
$\mathsf{R} \sqsubseteq \mathsf{S}$	$\mathsf{R}^{\mathcal{I}} \subseteq \mathsf{S}^{\mathcal{I}}$
$\mathsf{C} \sqsubseteq \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$

Assertional Axioms	
Syntax	Satisfied if
$\mathsf{a} : \mathsf{C}$	$\mathsf{a}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$
$(\mathsf{a},\mathsf{b}) : \mathsf{R}$	$(\mathsf{a}^{\mathcal{I}}, \mathsf{b}^{\mathcal{I}}) \in \mathsf{R}^{\mathcal{I}}$
$\mathsf{a} \doteq \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} = \mathsf{b}^{\mathcal{I}}$
$\mathsf{a} \neq \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} \neq \mathsf{b}^{\mathcal{I}}$

Figure 2: Syntax and Semantics of $\mathcal{SHIQ}$ ($n, m \in \mathbb{N}$, $n > 1, m > 0$, $\|\cdot\|$ denotes set cardinality, S is a simple role, $\mathsf{S}^{\sharp}(a,\mathsf{C}) = \|\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in \mathsf{S}^{\mathcal{I}},\ b \in \mathsf{C}^{\mathcal{I}}\}\|$).

an *instance assertion*, $(\mathsf{a},\mathsf{b}) : \mathsf{R}$ a *role assertion*, and $\mathsf{a} \doteq \mathsf{b}$ ($\mathsf{a} \neq \mathsf{b}$) an *individual equality (disjointness) assertion.*

A finite set of assertional axioms $\mathcal{A}$ w.r.t. a Tbox $\mathcal{T}_{\mathcal{R}}$ is called an *Abox*. If $(\mathsf{a},\mathsf{b}) : \mathsf{R} \in \mathcal{A}$ the a is called a R-predecessor of b and b a R-successor (or role filler) of a. An Abox $\mathcal{A}$ is *consistent* iff there exists an interpretation $\mathcal{I}$ which satisfies all assertions in $\mathcal{A}$ and all axioms in $\mathcal{T}$ w.r.t. $\mathcal{R}$.

2.1.2 Individual Pseudo Model Merging

In this part we review the individual pseudo model merging technique [67] for sake of completeness and readability. The technique of using an individual model merging test (see [67]) is based on the observation that individuals are usually members of only a small number of concepts and the Aboxes resulting from instance testing are proven as consistent in most cases. This was the motivation for devising the *individual pseudo model merging* technique. The basic idea is to have a fast, *sound* but possibly incomplete test for a focused individual i and a concept term $\neg\mathsf{D}$ without the need to explicitly consider role and concept assertions for all other individuals occurring in $\mathcal{A}$. These possible "interactions" are reflected in the definition of an "individual pseudo model" of i (see below).

For instance, in the DL $\mathcal{ALC}$ a pseudo model for an individual i mentioned in a consistent initial Abox $\mathcal{A}$ w.r.t. a Tbox $\mathcal{T}$ is defined as follows. Since $\mathcal{A}$ is consistent, there exists a set of completions $\mathcal{C}$ of $\mathcal{A}$. Let $\mathcal{A}' \in \mathcal{C}$. An *individual pseudo model* M for an individual i in $\mathcal{A}$ is defined as the tuple $\langle M^{\mathsf{D}}, M^{\neg\mathsf{D}}, M^{\exists}, M^{\forall}\rangle$ w.r.t. $\mathcal{A}'$ and $\mathcal{A}$ using the following definition.

$$
\begin{aligned}
M^{\mathsf{A}} &= \{\mathsf{A} \mid \mathsf{i} : \mathsf{A} \in \mathcal{A}',\ \mathsf{A} \text{ is a concept name}\} \\
M^{\neg\mathsf{A}} &= \{\mathsf{A} \mid \mathsf{i} : \neg\mathsf{A} \in \mathcal{A}',\ \mathsf{A} \text{ is a concept name}\} \\
M^{\exists} &= \{\mathsf{R} \mid \mathsf{i} : \exists\mathsf{R}.\mathsf{C} \in \mathcal{A}'\} \cup \{\mathsf{R} \mid (\mathsf{i},\mathsf{j}) : \mathsf{R} \in \mathcal{A}\} \\
M^{\forall} &= \{\mathsf{R} \mid \mathsf{i} : \forall\mathsf{R}.\mathsf{C} \in \mathcal{A}'\}
\end{aligned}
$$

The pseudo model of a concept D is defined analogously by using the completion of an initial Abox $\mathcal{A} = \{\mathsf{i}:\mathsf{D}\}$. Note the distinction between the initial Abox $\mathcal{A}$ and its completion $\mathcal{A}'$. Whenever a role assertion exists, which specifies a role successor for the individual i in the initial Abox, the referenced role name is added to the set $M^{\exists}$. This is based on the rationale that the cached pseudo model of i should not refer to individual names occurring already in the initial Abox $\mathcal{A}$. However, it is sufficient to reflect a role assertion $(\mathsf{i},\mathsf{j}):\mathsf{R} \in \mathcal{A}$ by adding the role name R to $M^{\exists}$. This guarantees that possible interactions via the role R are reflected. The function $pmodels\_mergable?$ is defined in Algorithm 1.

Algorithm 1 $pmodels\_mergable?(M_1, M_2)$

return $atoms\_mergable(M_1, M_2) \wedge roles\_mergable(M_1, M_2)$

The algorithm $atoms\_mergable$ tests for a possible concept interaction between a pair of pseudo models. It is applied to these pseudo models and returns *false* if $(M_1^{\mathsf{D}} \cap M_2^{\neg\mathsf{D}}) \neq \emptyset$ or $(\mathsf{M}_1^{\neg\mathsf{D}} \cap \mathsf{M}_2^{\mathsf{D}}) \neq \emptyset$. Otherwise it returns $true$.

The algorithm $roles\_mergable$ tests for a possible role interaction between a pair of pseudo models. It returns *false* if $(\mathsf{M}_1^{\exists} \cap \mathsf{M}_2^{\forall}) \neq \emptyset$ or $(\mathsf{M}_1^{\forall} \cap \mathsf{M}_2^{\exists}) \neq \emptyset$. Otherwise it returns $true$. The reader is referred to [67] for the proof of the soundness of this technique and for further details.

The algorithm $ind\_model\_merge\_poss?$ (used in the following sections) can be reduced to a pseudo model merging test as follows. It is assumed that $imodel(\mathsf{i}, \mathcal{A})$ returns the pseudo model of individual i w.r.t. $\mathcal{A}$ and $cmodel(\mathsf{D})$ the pseudo model of concept D.

Algorithm 2 $ind\_model\_merge\_poss?(\mathsf{i}, \mathsf{D}, \mathcal{A})$:

return $pmodels\_mergable?(imodel(\mathsf{i}, \mathsf{A}), cmodel(\mathsf{D}))$

2.1.3 GCI Absorption and Lazy Unfolding

Another standard optimization technique, GCI absorption [78, 77], tries to transform GCIs in a satisfiability preserving way in order to facilitate the application of the lazy unfolding technique [9]. In general, a Tbox can be divided into two sets of axioms. The set $\mathcal{T}_U$ contains axioms of the form $\mathsf{A} \sqsubseteq \mathsf{C}$ or $\neg\mathsf{A} \sqsubseteq \mathsf{D}$, where A is a concept name and C and D are concept expressions. The concept names occurring on the left-hand size of the axioms in $\mathcal{T}_U$ are called *unfoldable*. The second set $\mathcal{T}_G$ contains axioms of the form $\mathsf{C} \sqsubseteq \mathsf{D}$ where C and D are concept expressions (see [77] for a more detailed analysis of GCI absorption). The *lazy unfolding* technique works as follows. Whenever a tableau procedure encounters in an Abox an assertion of the form $\mathsf{a}:\mathsf{A}$ ($\mathsf{a}:\neg\mathsf{A}$) for the first time and an axiom of the form $\mathsf{A} \sqsubseteq \mathsf{C}$ ($\neg\mathsf{A} \sqsubseteq \mathsf{D}$) can be found in $\mathcal{T}_U$, it adds $\mathsf{a}:\mathsf{C}$ ($\mathsf{a}:\mathsf{D}$) to the Abox. The GCI absorption technique tries to maximize the efficacy of lazy unfolding by transforming axioms in $\mathcal{T}_G$ in such a way that they can be moved to $\mathcal{T}_U$. If possible, the set $\mathcal{T}_G$ should become empty after the application of GCI absorption. If axioms remain in $\mathcal{T}_G$, they have to be considered as possible disjunctions for every individual encountered during a satisfiability test (again, see [77] for more details). The number of remaining axioms in $\mathcal{T}_G$ is usually a good indication for the hardness of a given Tbox. Some of the optimization techniques presented in the following sections rely on subsumption tests which might become expensive if $\mathcal{T}_G$ is not empty after the absorption preprocessing step.

2.2 Optimization Techniques for Instance Retrieval

For applications, which either generate Aboxes on the fly as part of their problem-solving processes and/or ask a few queries w.r.t. an Abox, computing index structures by realization might not be worth the effort. Thus, in this section we discuss answering strategies for instance retrieval which is not based

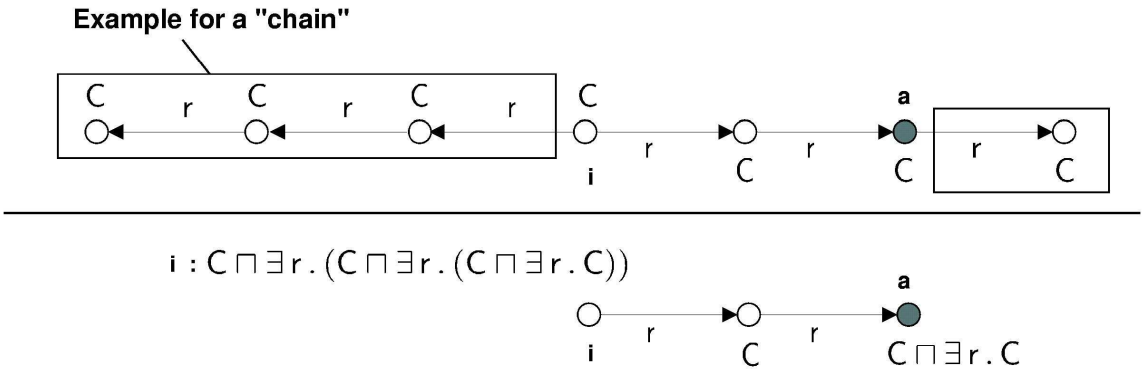

Figure 3: An example for contracting an Abox. The Abox part in the rectangle is replaced with a concept assertion (see the lower part for the resulting Abox).

on realization. However, as we will see later, these techniques can also be exploited if index structures are to be computed (possibly off-line).

2.2.1 Transformation of Aboxes

In order to make realization as fast as possible we investigated ways to maximize the effect of caching techniques supplied by RACER's Abox consistency checking architecture. We transform the original Abox in such a way that acyclic "chains" formed by role assertions are represented by an appropriate exists restriction (see [65] for a formal definition of the transformation rules). The corresponding concept and role assertions representing the chains are deleted from the Abox. We illustrate this contraction idea by an example presented in Figure 3. The idea is to transform tree-like role assertions (or "chains") starting from the individuals i and a into assertions with existential restrictions such that an equisatisfiable Abox is derived (see [65] for details). This transformation is similar to rolling-up techniques developed for conjunctive query answering (e.g., [56]). The contraction might be useful in situations where RACER supports caching of the satisfiability status of existential concept expressions (see also [65] for a discussion of a sound caching technique). Contracting an Abox is part of the process to build internal data structures for Abox reasoning.

2.2.2 Linear Instance Retrieval

One possible alternative for implementing instance retrieval is to consider one individual at a time. Hence, the standard Abox inference service $instance\_retrieval(\mathsf{C_q}, \mathcal{A})$ can be implemented by using the following procedure call: $linear\_retrieval(\mathsf{C_q}, contract(\mathsf{i}, \mathcal{A}), individuals(\mathcal{A}))$, where $contract(\mathsf{i}, \mathcal{A})$ computes the contracted variant of Abox $\mathcal{A}$ w.r.t. the individual i and $individuals(\mathcal{A})$ returns the set of individuals mentioned in Abox $\mathcal{A}$. Except for the contraction idea, linear instance retrieval was also implemented in a similar way in first generation DL systems (see, e.g., [117]).

We assume that SAT ($ASAT$) is the standard concept (Abox) satisfiability test implemented by an optimized tableau calculus [75, 62]. The function $linear\_retrieval$ is specified by Algorithm 3.

Algorithm 3 $linear\_retrieval(\mathsf{C}, \mathcal{A}, candidates)$:

```
result := {}
for all ind ∈ candidates do
    if instance?(ind, C, A) then
        result := result ∪ {ind}
    end if
end for
return result
```

2.2.3 Obvious Non-Instances: Exploiting Completion Information

The function call $instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$ could be implemented in a standard way as $\neg ASAT(\mathsf{A} \cup \{\mathsf{i} : \neg\mathsf{C}\})$. However, although this implementation of $instance?$ is sound and complete, it is quite inefficient. A faster variant uses sound but incomplete initial tests for detecting "obvious" non-instances: the individual pseudo model merging test (see [67] and Section 2.1.2) and possibly a subsumption test involving the negation of the query concept. These two tests (also referred to as "guards" for avoiding to invoke the tableau algorithm) are used in Algorithm 4.

Algorithm 4 $obv\_non\_instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$:

```
if ind_model_merge_poss?(i, negated_concept(C), A) then
    return true
else if use_subsumption_test_on_negated_concept then
    return subsumes?(negated_concept(C), individual_concept(i, A))
else
    return false
end if
```

The function $negated\_concept$ used in $obv\_non\_instance?$ returns the negation of its input concept whereas the function $individual\_concept$ returns for an individual i and an Abox $\mathcal{A}$ the conjunction of concepts occurring in all Abox concept assertions for i. If role assertions mentioning i are present in $\mathcal{A}$, additional concepts are added to i's individual concept. For role assertions of the form $(\mathsf{i}, \mathsf{j}) : \mathsf{R}$ and $(\mathsf{j}, \mathsf{i}) : \mathsf{R}$ either concepts of the form $(\exists_{\geq 1}\, \mathsf{R})$ and $(\exists_{\geq 1}\, \mathsf{R}^-)$ are added or, provided the unique name assumptions holds, $\exists_{\geq n}\, \mathsf{R}$ and $\exists_{\geq n}\, \mathsf{R}^-$ are generated by $individual\_concept$ where n depends on the number of different individual names j_k occurring in $(\mathsf{i}, \mathsf{j}_k) : \mathsf{R}$ and $(\mathsf{j}_k, \mathsf{i}) : \mathsf{R}$.

Algorithm 5 $subsumes?(\mathsf{C}, \mathsf{D})$:

```
if pmodels_mergable?(cmodel(negated_concept(C)), cmodel(D)) then
    return false
else
    return ¬SAT(D ⊓ ¬C)
end if
```

The subsumption test $subsumes?$ is implemented in Algorithm 5. Experience has shown that for some KBs a SAT test with an individual concept is often faster compared to a corresponding $ASAT$ (although both are of the same worst-case complexity class). The reason is that better optimization techniques are available for SAT than for $ASAT$ tests. However, if the GCIs contained in an KB are "difficult", i.e., enforce a high amount of runtime, the $subsumes?$ test in Algorithm 4 might become too expensive. A good indication seems to be if the set $\mathcal{T}_G$ of Tbox axioms which could not be absorbed is not empty (see Section 2.1.3). In general, our findings suggest to initialize $use\_subsumption\_test\_on\_negated\_concept$ with $false$.

Let us turn back to the function $obv\_non\_instance?$ now. If one of the "guards" in $obv\_non\_instance?$ returns $true$, the result of $instance?$ is $false$. Although the $obv\_non\_instance?$

test is often successful, we devised another non-instance test reusing the saved completion information. The function $completion\_ASAT$ (see Algorithm 6) invokes $ASAT$ but uses a saved completion $\mathcal{A}'$ instead of the original input Abox $\mathcal{A}$. If $completion\_ASAT$ is unsuccessful, an "expensive" instance test using the original Abox $\mathcal{A}$ is performed as shown in Algorithm 7.

Algorithm 6 $completion\_ASAT(i, C, A)$

return $ASAT(completion(A) \cup \{i : \neg C\})$

The function $completion(\mathcal{A})$ returns an associated completion $\mathcal{A}'$ for an Abox $\mathcal{A}$. The completion technique is sound but incomplete because if the completion $\mathcal{A}'$ extended by the assertion $\mathsf{i} : \neg \mathsf{C}$ is satisfiable, then the individual i is obviously not an instance of C. However, if the extended completion $\mathcal{A}'$ is unsatisfiable, an $ASAT$ test, where the original input Abox $\mathcal{A}$ is extended, might still find a different completion $\mathcal{A}''$ that also satisfies the assertion $\mathsf{i} : \neg \mathsf{C}$.

With these auxiliaries, the function $instance?$ can be optimized for the average case but is still sound and complete.

Algorithm 7 $instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$:

if $obv\_non\_instance?(i, C, A) \vee completion\_ASAT(i, C, A)$ **then**
  return *false*
else
  return $\neg ASAT(\mathsf{A} \cup \{\mathsf{i} : \neg \mathsf{C}\})$
end if

Although for many queries the result often consists of a small set of individuals, other individuals might still cause the "expensive" $ASAT$ test to be invoked, regardless of the "guards" in Algorithm 7. Thus, the number of $ASAT$ test should be further reduced in order to further improve the performance of instance retrieval.

2.2.4 Obvious Instances: Exploiting Precompletion Information

Another central optimization technique to ensure data description scalability – required for ontologies such as LUBM – is to also find "obvious" instances with minimum effort. Given an initial Abox consistency test and a completion one can consider all deterministic restrictions, i.e., one considers only those completion data structures (from now on called constraints) for which there are no choice points in the tableau proof (in other words, one considers only those constraints that do not have or-dependency information attached). These constraints constitute a so-called precompletion. Note that in a precompletion, no restrictions are violated because we assume that the precompletion is computed from an existing completion.

Given the precompletion constraints, for an individual i an approximation of its most-specific concept (MSC) is computed (the approximation is called MSC'). The generation of MSC' is identical to the generation of an individual concept (see the previous section) but is is based on a precompletion instead of a completion. If MSC'_{i} is subsumed by a query concept C, then i must be an instance of C. In the case of LUBM many of the assertions lead to deterministic constraints in the tableau proof which, in turn, results in the fact that for many instances of a query concept C (e.g., $\mathsf{Faculty}$ as in query $Q9$) the instance problem is decided with a subsumption test based on the MSC' of each individual. Subsumption tests are known to be usually fast due to caching and model merging. The more precisely MSC'_{i} approximates MSC_{i}, the more often an individual can be determined to be an obvious instance of a query concept. Obviously, it might be possible to determine obvious instances by directly considering the precompletion data structures. However, at this implementation level a presentation would be too detailed. The main

point is that, due to our findings, the crude approximation with MSC' suffices to solve many instance tests in KBs such as LUBM.

If atomic concepts are used as testers in the case of LUBM, a large number of tests for obvious non-instances or obvious instances can determine the result. However, for some individuals (e.g., i) and query concepts (e.g., C) both tests might not determine whether i is (not) an instance of C (e.g., this is the case for Chair). Since both of these "cheap" tests are incomplete, for these individuals (e.g., i) a refutational Abox consistency test where the assertion $\mathsf{i}:\neg\mathsf{C}$ has been added to the Abox must be decided with a sound and complete tableau prover. For some concepts, the set of candidates might become quite large. Considering the volume of assertions in LUBM , it is easy to see that the Abox consistency test should not start from the initial, unprocessed Abox in order to ensure scalability.

For large Aboxes and many repetitive instance tests it is a waste of resources to "expand" the very same initial constraints over and over again. Therefore, the precompletion resulting from the initial Abox consistency test is used as a starting point for refutational instance tests. The tableau prover keeps the precompletion in memory. All deterministic constraints have been already expanded, so, if some constraint is added, only a limited amount of work needs to be done.

2.2.5 Binary Instance Retrieval

How can Abox satisfiability tests be avoided at all? The observation is that only very few additions to $\mathcal{A}$ of the kind $\{\mathsf{i}:\neg\mathsf{C}\}$ lead to an inconsistency in the function $instance?$ (i.e., in very few situations i is indeed an instance of C). Thus, it might advantageous to combine several individual (non-)instance tests into one Abox consistency test based on a standard divide-and-conquer strategy. This scheme is based on splitting a set of individuals into binary partitions for instance retrieval as shown in Algorithm 8.

Algorithm 8 $binary\_retrieval(\mathsf{C}, \mathcal{A}, candidates)$:

```
if candidates = ∅ then
    return ∅
else
    ⟨partition_1, partition_2⟩ := partition(candidates)
    return partition_retrieval(C, A, partition_1, partition_2)
end if
```

We assume now that $instance\_retrieval(\mathsf{C_q}, \mathcal{A})$ is implemented by calling the procedure $binary\_retrieval(\mathsf{C_q},\ contract(\mathsf{i}, \mathcal{A}), individuals(\mathcal{A}))$. The function $partition$ is defined in Algorithm 9. It divides a set into two partitions of approximately the same size. Given the partitions, $binary\_retrieval$ calls $partition\_retrieval$. The idea of $partition\_retrieval$ (see Algorithm 11) is to first check whether *none* of the individuals in a partition is an instance of the query concept C. This is done with the function $non\_instances?$ (see Algorithm 10).

Algorithm 9 $partition(s)$: /* $s[i]$ refers to the i^{th} element of the set s */

```
if |s| ≤ 1 then
    return ⟨s, ∅⟩
else
    return ⟨{s[1], ..., s[⌊n/2⌋]}, {s[⌊n/2⌋ + 1], ..., s[n]}⟩
end if
```

Algorithm 10 $non\_instances?(cands, \mathsf{C}, \mathcal{A})$:

```
return ASAT(A ∪ {i:¬C | i ∈ cands ∧ ¬obv_non_instance?(i, C, A)})
```

The potential performance gain is based on the observation that the $non\_instances?$ test is successful in many cases. Hence, with one "expensive" Abox test a large set of candidates can be eliminated. The underlying assumption is that, in general, the computational costs of checking whether an Abox $(\mathcal{A} \cup \{\mathsf{i}:\neg\mathsf{C},\ \mathsf{j}:\neg\mathsf{C}, \ldots\})$ is consistent is largely dominated by $\mathcal{A}$ alone. Hence, it is assumed that the size of the set of constraints added to $\mathcal{A}$ has only a limited influence on the runtime. For knowledge bases with, for instance, cyclic GCIs or a non-empty set $\mathcal{T}_G$ of unabsorbed GCIs, this may not be the case, however. Partitioning a set of candidates in two parts of approximately the same size can be controlled by heuristics. This has not yet been fully explored. Thus, further performance gains might be possible.

Algorithm 11 $partition\_retrieval(\mathsf{C}, \mathcal{A}, part_1, part_2)$:

```
if |part_1| = 1 then
    let i be the only member of part_1
    if instance?(i, C, A) then
        return {i} ∪ binary_retrieval(C, A, part_2)
    else
        return binary_retrieval(C, A, part_2)
    end if
else if non_instances?(part_1, C, A) then
    return binary_retrieval(C, A, part_2)
else if non_instances?(part_2, C, A) then
    return binary_retrieval(C, A, part_1)
else
    return binary_retrieval(C, A, part_1) ∪ binary_retrieval(C, A, part_2)
end if
```

2.2.6 Dependency-based Instance Retrieval

Although $binary\_retrieval$ is found to be faster than linear retrieval in the average case, one can do better. If the function $non\_instances?$ returns $false$, one can analyze the dependencies of the tableau structures ("constraints") involved in all clashes of the tableau branches. Analyzing dependency information for a clash reveals the "original" Abox assertions responsible for the clash. If the clashes in all attempts to construct a completion are due to an added constraint $\mathsf{i}:\neg\mathsf{C}$, then, as a by-product of the test, the individual i is known to be an instance of the query concept C. The individual can be eliminated from the set of candidates to be investigated, and it is definitely part of the solution set. Dependency information is kept for other optimization purposes as well [78] and dependency analysis does not involve much overhead.

Eliminating candidate individuals detected by dependency analysis prevents the reasoner from detecting the same clash over and over again until a partition of cardinality 1 is tested. If the solution set is large compared to the set of individuals in an Abox, there is some overhead compared to linear instance retrieval because only one individual is removed from the set of candidates at a time as well with the additional cost of collecting dependency information during the tableau proofs.

2.3 Static Index-based Instance Retrieval

The techniques introduced in the previous sections can also be exploited if indexing techniques are used for instance retrieval (see, e.g., [117, p. 108f]). Basically, the idea is to reduce the set of candidates that have to be tested by computing the direct types of every individual. An index is constructed by deriving a function $associated\_inds$ defined for each concept name C mentioned in the Tbox such that $\mathsf{i} \in associated\_inds(\mathsf{C})$ iff $\mathsf{C} \in direct\_types(\mathsf{i}, \mathcal{A})$. The optimizations used in RACER are inspired by

the marking and propagation techniques described in [15, 14] for exploiting explicitly given information as much as possible.

In the following we assume that CN is the set of all concept names mentioned in a given Tbox (including the name $\top$). Furthermore, it is assumed that the function $children(\mathsf{C})$ ($parents(\mathsf{C})$) returns the least specific subsumees (most specific subsumers) of C whereas the function $descendants(\mathsf{C})$ ($ancestors(\mathsf{C})$) returns all subsumees (subsumers) of C. The descendants and ancestors of C include C. Subsumers and subsumees of a concept C are concept names from CN. The function $synonyms(C)$ returns all concept names from CN which are equivalent to C.

The standard way to compute the index is to compute the direct types for each individual mentioned in the Abox separately (one-individual-at-a-time approach). In order to compute the direct types of individuals w.r.t. a Tbox and an Abox, the Tbox must be classified first. Static index-based instance retrieval was investigated in [117, p. 108f.] and is implemented as follows.

Algorithm 12 $static\_index\_based\_retrieval(\mathsf{C}, \mathcal{A})$:

if $\exists\, \mathsf{N} \in CN : \mathsf{N} \in synonyms(\mathsf{C})$ **then**
  return $\bigcup_{\mathsf{D} \in descendants(\mathsf{C})} associated\_inds(\mathsf{D})$
else
  $known := \bigcup_{\mathsf{D} \in descendants(\mathsf{C})} associated\_inds(\mathsf{D})$
  $candidates := \bigcup_{\mathsf{P} \in parents(\mathsf{C})} \bigcup_{\mathsf{D} \in descendants(\mathsf{P})} associated\_inds(\mathsf{D})$ (*)
  return $known \cup instance\_retrieval(\mathsf{C}, \mathcal{A}, candidates \setminus known)$
end if

It is obvious that $instance\_retrieval$ can be implemented by any of the techniques introduced in the previous sections.

Computing the index structures (i.e., the function $associated\_inds$) is known to be time-consuming. Our findings indicate that for many applications this might take several minutes or even hours, i.e. index computation is only possible in a setup phase. Since for many applications this is not tolerable, new techniques had to be developed. The main problem is that for computing the index structure $associated\_inds$ the direct types are computed for every individual in isolation. Rather than looping over all individuals and asking for the direct types of each individual in a separate query, we investigated the idea of using *sets* of individuals which are "sieved" into the taxonomy.

We call this approach the sets-of-individuals-at-a-time approach (see Algorithms 13 and 14). The traverse procedure (Algorithm 13) sets up the index has_member. For a given concept name A the function has_member returns the set of A's known instances. The procedure $compute\_index\_sets\_of\_inds\_at\_a\_time$ (Algorithm 14) uses has_member in order to check if the instances of a concept name are not instances of the children of the concept name. Being this the case, the concept name is marked as one of the direct types of each of the instances. This is done by setting up the index $associated\_inds$ appropriately.

Algorithm 13 $traverse(inds, \mathsf{C}, \mathcal{A}, has\_member)$:

if $inds \neq \emptyset$ **then**
  for all $\mathsf{D} \in children(\mathsf{C})$ **do**
    if $has\_member(\mathsf{D}) = \text{unknown}$ **then**
      $instances\_of\_D := instance\_retrieval(\mathsf{D}, inds, \mathcal{A})$
      $has\_member(\mathsf{D}) := instances\_of\_D$
      $traverse(instances\_of\_D, \mathsf{D}, \mathcal{A}, has\_member)$
    end if
  end for
end if

Algorithm 14 *compute_index_sets_of_inds_at_a_time*($\mathcal{A}$):

```
for all C ∈ CN do
    has_member(C) := unknown
    associated_inds(C) := ∅
end for
traverse(individuals(A), top, A, has_member)
has_member(top) := individuals(A)
for all C ∈ CN do
    if has_member(C) ≠ unknown then
        for all ind ∈ has_member(C) do
            if ¬∃D ∈ children(C) : ind ∈ has_member(D) then
                associated_inds(C) := associated_inds(C) ∪ {ind}
            end if
        end for
    end if
end for
```

2.3.1 Dynamic Index-based Instance Retrieval

Computing a complete index (realization) as described in the previous subsection is possible if many queries are posed w.r.t. a "fixed" Abox (and Tbox). However, sometimes realization is too time-consuming. Therefore, we investigated a new strategy that exploits (i) explicitly given information (e.g., from Abox assertions of the form $\mathsf{i} : \mathsf{A}$ where A is a concept name) and (ii) the results of previous instance retrieval queries.

The idea can be explained as follows. The function *associated_inds* associates a set *Inds* of individuals with each concept name C such that for each $\mathsf{i} \in Inds$ it holds that i is an instance of C, for each $\mathsf{D} \in descendants(\mathsf{C})$ the individual $\mathsf{i} \notin associated\_inds(\mathsf{D})$, and for each $\mathsf{D} \in ancestors(\mathsf{C})$ the individual $\mathsf{i} \notin associated\_inds(\mathsf{D})$.

The function *associated_inds* is updated due to the results of queries. Let us assume $\mathsf{i} \in associated\_inds(\mathsf{C})$ and $\mathsf{C} \in ancestors(\mathsf{E})$. If it turns out that i is an instance of E, the function *associated_inds* is changed accordingly. Thus, the index evolves as instance retrieval queries are answered. Therefore, we call this strategy dynamic index-based instance retrieval.

In this new approach, the function $associated\_inds(\mathsf{C})$ returns an individual i even if C is not "most specific", i.e., even if there might exist a subconcept D of C such that i is also an instance of D. The consequence is that Algorithm 12 is no longer complete. The idea of only considering the parents of the query concept (see the line marked with an asterisk in Algorithm 12) must be dropped. Before we give a complete algorithm for dynamic index-based instance retrieval, further optimization techniques are introduced.

Let us assume concept D is a subsumer of C. In addition, let us assume in order to answer some previous query the direct types for an individual i are computed. If it is known for an individual $\mathsf{i} \in associated\_inds(\mathsf{D})$ that $\mathsf{D} \in direct\_types(\mathsf{i})$, then i is removed from the set of candidates for the query concept C. Since D is a subsumer of C and D is a direct type (i.e., D is most specific), i cannot be an instance of C.

With each concept name we also associate a set of non-instances. The non-instances are found by queries for the direct types of an individual (the non-instances are associated with the children of each direct type) or by exploiting previous calls to the function *instance_retrieval*. If an individual i is found not to be an instance of a query concept D, this is recorded appropriately by including i in $associated\_non\_instance(\mathsf{D})$ if there is no $\mathsf{E} \in ancestors(\mathsf{D})$ such that $\mathsf{i} \in associated\_non\_instance(\mathsf{E})$ (non-redundant caching). The non-instances of a query concept can then be discarded from the set of candidates. The new algorithm for instance retrieval is shown in Algo-

rithm 15.

Algorithm 15 $dynamic\_index\_based\_retrieval(\mathsf{C}, \mathcal{A})$:

$known := \bigcup_{\mathsf{D} \in descendants(\mathsf{C})} associated\_inds(\mathsf{D})$
$possible\_candidates := \bigcup_{\mathsf{D} \in (ancestors(\mathsf{C}) \setminus \{\mathsf{C}\})} associated\_inds(\mathsf{D})$
$candidates := possible\_candidates \setminus \bigcup_{\mathsf{D} \in ancestors(\mathsf{C})} associated\_non\_instances(\mathsf{D})$
return $known \cup instance\_retrieval(\mathsf{C}, \mathcal{A}, candidates \setminus known)$

Note that instead of testing the parents as done in Algorithm 12 (see the line marked with an asterisk), in Algorithm 15 the descendants of the query concept C are taken into consideration for possible candidates. In other words, it is not a problem if an individual i is returned by $associated\_inds(\mathsf{D})$ although there exist subconcepts of D of which i is also an instance.

2.3.2 OWL-DL Datatype Properties

RACER supports concrete domain reasoning as defined in [62] and reasoning about fillers of OWL-DL datatype properties can be easily mapped on concrete domain reasoning using appropriate concrete domains for strings, booleans, and integers. However, this mapping turns out to be too expensive for datatype properties unless some of them are restricted by number restrictions. RACER analyzes submitted KBs and dynamically enables a special and simpler reasoning method for datatype properties if permitted. Fillers of datatype properties adhere to a locality restriction, e.g., if two different individuals i and j have fillers "*joe*" and "*bill"* for a datatype property has_name, the reasoning remains local w.r.t. i and j respectively. In other words: there cannot exist an interaction between the datatype property filler of i and j. This optimization technique seems to be quite effective for very large Aboxes such as LUBM containing many assertions about datatype properties.

2.3.3 Re-use of Role Assertions

Standard tableau methods for DLs avoid the expansion of assertions of the form $\mathsf{i} : \exists \mathsf{R}.\mathsf{C}$ if there already exists a R-role filler for i that is known to be an instance of concept C. This re-use of already existing role fillers is useful but might be expensive for large Aboxes if appropriate indexes for fast role filler lookup do not exist. RACER implements corresponding index structures enabling a role filler lookup in almost constant time. This technique is advantageous for very large Aboxes such as UOBM.

2.4 Optimizations for Grounded Conjunctive Queries

LUBM queries are modeled as grounded conjunctive queries referencing concept, role, and individual names from the Tbox. Below, LUBM queries 9 and 12 are shown in order to demonstrate LUBM query answering problems – note that 'www.University0.edu' is an individual and $subOrganizationOf$ is a transitive role. Please refer to [58, 59] for more information about the LUBM queries.

$$\begin{aligned} Q9 : ans(x, y, z) \leftarrow\ & Student(x), Faculty(y), Course(z), \\ & advisor(x, y), takesCourse(x, z), teacherOf(y, z) \\ Q12 : ans(x, y) \leftarrow\ & Chair(x), Department(y), memberOf(x, y), \\ & subOrganizationOf(y, \text{www.University0.edu}) \end{aligned}$$

In order to investigate the data description scalability problem, we used a Tbox provided with the LUBM benchmarks. Tboxes uses inverse and transitive roles as well as domain and range restrictions, but no number restrictions, value restrictions or disjunctions. Among other axioms, the LUBM Tbox contains axioms that express necessary and sufficient conditions for some concept names. For instance, the Tbox contains an axiom for Chair: $\mathsf{Chair} \equiv \mathsf{Person} \sqcap \exists \mathsf{headOf}.\mathsf{Department}$.

If grounded conjunctive queries are answered in a naive way by evaluating subqueries in the sequence of syntactic notation, acceptable answering times can hardly be achieved. For efficiently answering queries, a query execution plan is determined by a cost-based optimization component (c.f., [54, p. 787ff.], see also [45]) which orders query atoms such that queries can be answered effectively. Query execution plans are specified in the same notation as queries (whether a query is seen as an execution plan will be clear from context). We assume that the execution order of atoms is determined by the order in which they are textually specified.

Let us consider the execution plan $ans(x, y) \leftarrow C(x), R(x, y), D(y)$. Processing the atoms from left to right will start with the atom $C(x)$. Since there are no bindings known for the variable x, the atom $C(x)$ is mapped into an instance retrieval query $instance\_retrieval(C, \mathcal{A})$. The elements in the result set of the retrieval query are possible bindings for x. $C(x)$ is called a *generator*. The next query atom in the execution plan is $R(x, y)$. There are bindings known for x but no bindings for y. Thus, $R(x, y)$ is also a generator (for y-bindings). Given the atom $R(x, y)$ is handled by a role filler query for each binding of x, there are possible bindings for y generated. Afterwards, the atom $D(y)$ is treated. Since there are bindings for y available, the atom is mapped to an instance test (for each binding). We say, the atom $D(y)$ acts as a *tester*.

Determining all bindings for a variable (with a generator) is much more costly than verifying a particular binding (with a tester). Treating the one-place predicates $Student$, $Faculty$, and $Course$ from query Q9 (see above) as generators for bindings for corresponding variables results in combinatorial explosion (cross product computation). Optimization techniques are required that provide for efficient query answering in the average case.

2.4.1 Query Optimization

The optimization techniques that we investigated are inspired by database join optimizations, and exploit the fact that there are few $Faculties$ but many $Students$ in the data descriptions. For instance, in case of query Q9, the idea is to use $Faculty$ as a generator for bindings for y and then generate the bindings for z following the role $teacherOf$. The heuristics applied here is that the average cardinality of a set of role fillers is rather small. For the given z bindings we apply the predicate $Course$ as a tester (rather than as a generator as in the naive approach). Given the remaining bindings for z, bindings for x can be established via the inverse of $takesCourse$. These x bindings are then filtered with the tester $Student$.

If z was not mentioned in the set of variables for which bindings are to be computed (in the head of the query), and the tester $Course$ was not used, there would be no need to generate bindings for z at all. One could just check for the existence of a $takesCourse$ role filler for bindings w.r.t. x. This way, further optimization is possible.

In the second example, query Q12, the constant (individual) named $www.University0.edu$ is mentioned. Starting from this individual the inverse of the role $subOrganizationOf$ is applied as a generator for bindings for y which are filtered with the tester $Department$. With the inverse of $memberOf$, bindings for x are computed which are then filtered with $Chair$. Since for the concept $Chair$ sufficient conditions are declared in the Tbox, instance retrieval reasoning is required if $Chair$ is a generator. Thus, it is advantageous that $Chair$ is applied as a tester (and only instance tests are performed).

For computing a query execution plan, a total order relation on query atoms with respect to a given set of data descriptions (assertions in an Abox) is required. For determining the order relation, we need information about the number of instances of concept and role names. An estimate for this information can be computed in a preprocessing step by considering given data descriptions, or could be obtained by examining the result set of previously answered queries (we assume that Abox realization is too costly, so this alternative is ruled out).

In Section 2.3 we have discussed that it is advantageous to compute an index $associated\_inds$ that allows us to find "obvious" instances by exploiting precompletion information. The index $associated\_inds$ is organized in such a way that retrieving the instances of a concept A, or one of its ancestors, requires (almost) constant time (in combination with the descendants, see Algorithm 12). This kind of index is particularly useful to provide bindings for variables if, despite all optimization attempts for deriving query execution plans, concept names must be used as generators. In addition, the index is used to estimate the cardinality of concept extensions. The estimates are used to compute an order relation for query atoms. The less the cardinality of a concept or a set of role fillers is assumed to be, the more priority is given to the query atom. Optimizing LUBM query $Q9$ with the techniques discussed above yields the following query execution plan.

$$\begin{aligned} Q9' : ans(x,y,z) \leftarrow\ & Faculty(y), teacherOf(y,z), Course(z), \\ & advisor^{-}(y,x), Student(x), takesCourse(x,z) \end{aligned}$$

Using this kind of rewriting, queries can be answered much more efficiently.

If the Tbox contains only GCIs of the form $\mathsf{A} \sqsubseteq \mathsf{A}_1 \sqcap \ldots \sqcap \mathsf{A}_\mathsf{n}$, i.e., if the Tbox forms a hierarchy, the index-based retrieval discussed in this section is complete (see [18]). However, this is not the case for LUBM. In LUBM, besides domain and range restrictions, axioms are also of the form $\mathsf{A} \doteq \mathsf{A}_1 \sqcap \mathsf{A}_2 \sqcap \ldots \sqcap \mathsf{A}_\mathsf{k} \sqcap \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \ldots \sqcap \exists \mathsf{R}_\mathsf{m}.\mathsf{B}_\mathsf{m}$ (actually, $\mathsf{m} = 1$). If sufficient conditions with exists restrictions are specified as in the case of $Chair$, optimization is much more complex. In LUBM data descriptions, no individual is explicitly declared as a $Chair$ and, therefore, reasoning is required, which is known to be rather costly. If $Chair$ is used as a generator and not as a tester such as in the simple query $ans(x) \leftarrow Chair(x)$, optimization is even more important. The idea to optimize instance retrieval is to detect an additional number of obvious instances by transforming sufficient conditions into conjunctive queries.

2.4.2 Transforming Sufficient Conditions into Conjunctive Queries

Up to now we can detect obvious instances based on told and taxonomical information (almost constant time, see the previous section) as well as information extracted from the precompletion (linear time w.r.t. the number of remaining candidate individuals and a very fast test, see Section 2.2.4). Known non-instances can be determined with model merging techniques applied to individual pseudo models (also a linear process w.r.t. the number of remaining candidate individuals but with a very fast test,

see Section 2.2.3). However, there might still be some candidates left. Using the results presented in Section 2.2 it is possible to use dependency-directed instance retrieval and binary partitioning. Our findings suggest that in the case of LUBM, for example for the concept $Chair$, the remaining refutational tableau proofs are very fast. However, for $Chair$ quite many candidates remain since there are many $Persons$ in LUBM. In application scenarios such as those we investigate with LUBM, we have 200 000 individuals and more with many $Persons$. Even if each single instance test lasts only a few dozen microseconds, query answering will be too slow, and hence additional techniques should be applied to solve the data description scalability problem.

The central insight for another optimization technique is that conjunctive queries can be optimized according to the above-mentioned arguments whereas for concept-based retrieval queries, optimization is much harder to achieve. Let us consider the query $ans(x) \leftarrow Chair(x)$. For $Chair$, sufficient conditions are given as part of the Tbox (see above). Thus, in principle, we are looking for instances of the concept $\mathsf{Person} \sqcap \exists \mathsf{headOf}.\mathsf{Department}$. The key to optimizing query answering becomes apparent if we transform the definition of $Chair$ into a conjunctive query and derive the optimized version $Q15'$:

$$Q15 : ans(x) \leftarrow Person(x), headOf(x, y), Department(y)$$
$$Q15' : ans(x) \leftarrow Department(y), headOf^{-}(y, x), Person(x)$$

Because there exist less $Departments$ than $Persons$ in LUBM, search for bindings for x is substantially more focused in $Q15'$ (which is the result of automatic query optimization, see above). In addition, in LUBM, the extension of $Department$ can be determined with simple index-based tests only (only hierarchies are involved).

In addition, there in the Tbox is a domain restriction $\mathsf{Professor}$ for the role headOf, which can be exploited to further optimize the query by making atoms as specific as possible. Due to the domain restriction for headOf, the variable x in Q15' must refer to a $\mathsf{Professor}$ instance, which can be made explicit. If we further exploit that $\mathsf{Professor}$ is subsumed by Person, it is clear that the atom $Person(x)$ can be dropped. We do not present the algorithm for determining the most-specific atoms w.r.t. a particular variable, though.

$$Q15'' : ans(x) \leftarrow Department(y), headOf^{-}(y, x), Professor(x)$$

With the $Chair$ example one can easily see that the standard approach for instance retrieval can be optimized dramatically with rewriting concept query atoms if certain conditions are met.

The idea of the transformation is to implement the inverse of the contraction or rolling-up technique (see [56] and also Section 2.2.1). Here, however, existential restrictions are "rolled-down" to conjunctive queries. The transformation is reminiscent of a transformation introduced in [23]. In this work, description logic concepts are translated to first-order logic formulae. The transformation approach discussed in this section is also reminiscent to a transformation approach discussed in [113]. Note, however, that in our approach, we additionally consider domain and range restrictions for roles in order to maximize information that can be used for exploiting for indexes.

Algorithm 16 $rewrite(tbox, concept, var)$:

```
if unabsorbed_gcis(tbox) ≠ ∅ ∨ definition(concept) = ⊤ then
   return (concept(var))
else
   {atom_1, ..., atom_n} := rewrite_0(tbox, concept, var, {})
     return (atom_1, ..., atom_n)
end if
```

The rewriting algorithm is defined in Algorithms 16, 17, and 18. Every concept query atom $C(x)$ used in a conjunctive query is replaced with $rewrite(query\_tbox, C, x)$ (and afterwards, the query is optimized

Algorithm 17 $rewrite\_0(tbox, concept, var, exp)$:

```
if definition(concept) = ⊤ ∨ concept ∈ exp then
    return {concept(var)}
else
    ;; catch installs a marker to which the control flow can be thrown
    catch not_rewritable
        rewrite_1(tbox, concept, definition(tbox, concept), var, {concept} ∪ exp)
end if
```

Algorithm 18 $rewrite\_1(tbox, concept\_name, definition, var, exp)$:

```
if (definition = A) where A is an atomic concept then
    return rewrite_0(tbox, definition, var, {definition} ∪ exp)
else
    if (definition = ∃R.C) then
        filler_var := fresh_variable()
        return {R(var, filler_var)} ∪ rewrite_0(tbox, C, filler_var, exp)
                ∪ rewrite_0(tbox, role_domain(R), var, exp)
                ∪ rewrite_0(tbox, role_range(R), filler_var, exp)
    else
        if (definition = C1 ⊓ ... ⊓ Cn) then
            return rewrite_1(tbox, concept_name, C1, var, exp)
                    ∪ ... ∪
                    rewrite_1(tbox, concept_name, Cn, var, exp)
        else
            ;; throw the control flow out of rewrite_1 recursion
            ;; back to the call to rewrite_1 in rewrite_0 and
            ;; return {concept_name(var)}
            throw not_rewritable {concept_name(var)}
        end if
    end if
end if
```

with the techniques describe above). If the transformation approach is applied to $Q15$, the query $Q15''$ is derived.

Some auxiliary functions are used. The function $definition(C)$ returns sufficient conditions for a concept name C (the result is a concept), and the function $unabsorbed\_gcis(tbox)$ indicates whether there are some unabsorbed GCIs left after GCI transformation, i.e., $\mathcal{T}_G \neq \emptyset$ (see Section 2.1.3, the result is a set of concepts). In addition, we use a function $fresh\_variable$ that generates a new variable that was not used before. The functions $role\_domain$ and $role\_range$ return the domain and range restrictions of a role (after GCI absorption).

If there is no specific definition or there are meta constraints, rewriting is not applied (see Algorithm 16). It is easy to see that the rewriting approach is sound. However, it is complete only under specific conditions, which can be automatically detected. If we consider the Tbox $\mathcal{T} = \{\mathsf{D} \equiv \exists \mathsf{R}.\mathsf{C}\}$, the Abox $\mathcal{A} = \{\mathsf{i} : \exists \mathsf{R}.\mathsf{C}\}$ and the query $ans(x) \leftarrow D(x)$, then due to the algorithm presented above the query will be rewritten as $ans(x) \leftarrow R(x, y), C(y)$. For variable bindings, the query language nRQL (see above) considers only those individuals that are explicitly mentioned in the Abox. Thus i not be part of the result set because there is no binding for y in the Abox $\mathcal{A}$. Examining the LUBM Tbox and Abox it becomes clear that in this case for every $\exists \mathsf{R}.\mathsf{C}$ which is applicable to an individual i there already exist constraints $(\mathsf{i}, \mathsf{j}) : \mathsf{R}$ and $\mathsf{j} : \mathsf{C}$ in the original Abox (LUBM was derived from a database schema). Existential restrictions are fulfilled by named individuals (see Section 2.3.3). However, even if this is not the case, the technique can be employed under some circumstances.

Usually, in order to construct a model (or a completion to be more precise), tableau provers create a

new individual for each constraint of the form $\mathsf{i} : \exists \mathsf{R}.\mathsf{C}$ and add corresponding concept and role assertions (if not already present). These newly created individuals are called anonymous individuals. Let us assume, during the initial Abox consistency test a completion is found. As we have discussed above, a precompletion is computed by removing all constraints that depend on a choice point. If there is no such constraint, the precompletion is identical to the completion that the tableau prover computed. Then, the set of bindings for variables generated by $fresh\_variable()$ is extended to the anonymous individuals found in the precompletion. The rewriting technique for concept query atoms is applicable (i.e., is complete) under these conditions. Even if the rewriting technique is not complete (because something has been removed from a completion to derive a precompletion), it can be employed to reduce the set of candidates for binary partitioning techniques that can speed of this process considerably in the average case.

3 An Architectural Framework for Building OBISs

The section is structured as follows. We first describe the overall framework and explain how the identified problems in the introduction (**P1–P7**) are addressed. Next we describe the substrate QL framework SUQL, which plays a crucial role in this work. The NRQL ABox QL [68, 69] is discussed as a concrete instantiation of SUQL.

In this section we first describe the framework from the knowledge level perspective [118]. From a logical point of view, a so-called *substrate data model* is introduced, and the main principles of the associated query language SUQL are presented. We also briefly remark on implementation aspects, the symbol level perspective. We believe that both perspectives on a reasoning system are of equal importance in order to guarantee empirical success. A "good design" should encompass both perspectives in order to avoid performance bottlenecks and impedance mismatches. After having presented the framework, we discuss how the problems **P1 – P7** are tackled. Please recall that **P1 – P7** provide the motivation for the whole approach; more precisely, **P1, P2** address the knowledge level, whereas the problems **P3 – P7** address the symbol level.

We do *not* claim that the substrate model is interesting from a theoretical perspective. Its generic character is of course also its weakness. Thus, it must be *specifically instantiated*. An *instantiation of the model* results in a specific substrate type, e.g. a substrate type *ABox*. The formalization presented here is only as detailed and formally elaborated as is beneficial and required for the description of the semantics of the services, especially of the query answering service. We claim that the presented formalization is sufficient for our purpose.

From the knowledge level perspective, the data model is partially inspired by the work on $\mathcal{E}$-Connections [94], tableaux data structures [74], as well as by RDF(S). However, it would be inappropriate to claim that this is an $\mathcal{E}$-Connection application, since we are basically just using labeled graphs, defined by means of first order logic, and similar knowledge models have been used in AI since the 1960s [8, Chapter 4] (although the substrate model is primarily an *extensional* knowledge model). SUQL is inspired by [47].

From the symbol level perspective, our approach is related to JENA [107], but we have a somewhat broader scope, and the underlying knowledge (data) models are more general than RDF(S), as will become clear in the following.

3.1 The Knowledge Level Perspective

Formally, we base our framework on a graph-based data model which provides the required flexibility and extensibility for the extensional component, the so-called *substrate data model*. A generic substrate query language called SUQL for this data model provides the required flexibility and extensibility on the QL side. The definition of SUQL is only prepared in this section and continued and elaborated on in Section 3.5.

The substrate model serves both as a mediator and as an abstraction layer ("semantic middleware"). It enables us to specify and build extensional representation layers for special-domain and hybrid representations and is sufficiently general to also encompass ABoxes and RDF(S) graphs. A substrate is thus defined as a very general notion:

Definition 1. A *substrate* is an edge- and node-labeled directed graph $(V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$, with V being the set of *substrate nodes*, and E being a set of *substrate edges*. The node labeling function $L_V : V \to \mathcal{L}_\mathcal{V}$ maps nodes to descriptions in an appropriate node description language $\mathcal{L}_\mathcal{V}$, and likewise for $L_E : E \to \mathcal{L}_\mathcal{E}$, where $\mathcal{L}_\mathcal{E}$ is an edge description language.

If $(i, j) \in E$, then j is called a *successor* of i, and i is called a *predecessor* of j. In case $R \in L_E((i, j))$, we can (more specifically) talk of an R-successor resp. -predecessor. △

The languages $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are not fixed and can be seen as *subsets of first-order predicate logic, FOPL, (denoted in variable-free syntax)*, e.g., some modal logic, description logic, or propositional logic. Using this FOPL perspective, V is a set of constant symbols, and L_V and L_E are *indexing functions* into sets of closed FOPL formulas.

Let us illustrate this with an example. Consider an $\mathcal{ALC}$ ABox $\mathcal{A}$. We can consider this ABox as a substrate $S = (V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$ if we identify V with the ABox individuals, $V = \mathsf{inds}(\mathcal{A})$, E with the set of pairs of individuals mentioned as arguments in role assertions, $E = \{\, (i,j) \mid\ (i,j) : R \in \mathcal{A} \,\}$, with $\mathcal{L}_\mathcal{V} = \mathcal{ALC}$, and $\mathcal{L}_\mathcal{E} = (\mathcal{N}_\mathcal{R}, \sqcap)$ would be the set of $\mathcal{ALC}$ role names $\mathcal{N}_\mathcal{R}$ closed under conjunction, such that $C \in L_V(i)$ iff $i : C \in \mathcal{A}$, and $R_1 \sqcap \cdots \sqcap R_n = L_E((i,j))$ iff $\{\, R_1, \ldots, R_n \mid\ R_i \in \mathcal{N}_\mathcal{R}, (i,j) : R_i \in \mathcal{A} \,\}$. From the FOPL perspective, $L_V(i)$ and $L_E((i,j))$ correspond to $\{\Phi(C)_{x \leftarrow i}, \ldots, R_1(i,j), \ldots, R_n(i,j)\}$, where $\Phi(C)$ returns the *FOPL standard translation* [8, pp. 50] of the concept C, which is a first order formula with one free variable, x, e.g. $\Phi(\exists R.C) = \exists y R(x,y) \wedge C(y)$.

However, for many substrates, the corresponding FOPL set will simply contain *ground atoms (facts)*.

An *associated TBox of an ABox* manifests itself in additional FOPL sentences. Formally, we simply define a *substrate with a background theory* (having an additional set of closed FOPL axioms). These additional FOPL sentences are obtained by applying the standard translation to the TBox axioms.

The SUQL framework allows for the definition of specialized substrate QLs, tailored for special substrate classes (e.g., ABoxes, SBoxes). The SUQL framework is based on the general notion of *(ground) query atom entailment*. All that matters here is that a *notion of logical entailment* between a substrate S and a *query atom for S* is defined and decidable. Query atoms are, conceptually slightly simplified, again FOPL formulas with one or two free FOPL variables (we use x and y in the remaining paper for these); the atoms are thus called unary (resp. binary) query atoms. Thus, $S \models P_{x \leftarrow i}$ must be decidable for the unary atom P and the node $i \in V$, and $S \models Q_{x \leftarrow i, y \leftarrow j}$ must be decidable for the binary atom Q and the nodes $i, j \in V$.

The SUQL framework provides a great deal of flexibility, extensibility and adaptability, since specialized query atoms (resp. P and Q) can be tailored for specific substrate classes.

3.2 The Symbol Level Perspective

A substrate is an instance of a CLOS (Common Lisp Object System) class [137] – a *substrate class* thus provides the implementation of a *substrate type (or kind).* On the one hand, a substrate is thus a representation on the knowledge level, but on the other hand also – and much more importantly in this work – a structure on the symbol (or implementation) level.

We have already used the phrase *instantiation of the substrate data model* informally. More specifically, from now on this means that a new substrate class is defined (tailored for certain representation tasks) by means of subclassing. In the same sense we are using the phrase SUQL *instantiation* to refer to a specialized substrate QL, e.g., one that offers substrate-specific, tailored query atoms. Last but not least, an *instantiation of the framework* encompasses all kinds of instantiations; for example, the DLMAPS system is an instantiation which contains specific substrate types and specialized SUQL instantiations.

Since CLOS offers multiple inheritance (i.e., allows a class to have multiple parent classes), it becomes possible to define *combinations of substrates.* For example, one can define a substrate class *spatial ABox* having the substrate classes *ABox* and *SBox* as parents. As a result, instances in such a spatial ABox are, on the one hand, *ABox individuals*, and instances of spatial datatypes on the other hand [157]. This can eliminate the need for a *hybrid representation* in favor of an *integrated* representation. However, the substrate data model also supports hybrid representations.

Another important idea is that the nodes and edges in a substrate can be "virtual", i.e., the substrate is simply used as a mediation layer or "facade" that provides a graph perspective on a different represen-

tation, e.g. a RACER ABox. In this case, the API functions of the substrate just pass through to the API functions of RACER. Thus, a substrate class *may or may not* correspond to a physical store.

Not only substrates, but also SUQL query atoms are instances of CLOS classes. This enables the definition of the $\models$ relation as a (binary) CLOS *multi-method* `substrate-entails-atom-p`. A multi-method is polymorphic (does late binding) according to the types of all its arguments [137], unlike languages like Java, where only the type of the first argument is used for dispatching. Thus, depending on the class of substrate and atom, different inference algorithms will be called for (e.g., a DL system API function in case an ABox is queried, and a geometric algorithm performing some kind of *spatial model checking* if an SBox is queried). Furthermore, *intrinsically encoded axioms* can be taken into account in the implementation of a `substrate-entails-atom-p` method, simply by means of programming. For example, the Clark completion axioms must not be explicitly present as sentences. They are only needed for a description of the semantics on the knowledge level, but not on the symbol level.[3]

The generic SUQL *query answering engine* immediately supports the evaluation of specialized atoms once a `substrate-entails-atom-p` method is applicable, since there are generic enumerator and tester methods defined. However, these will not exhibit good performance, since they only implement *linear retrieval algorithms* (instances are retrieved using "enumerate and test"). However, good performance can be achieved if *dedicated generators and tester* are defined for specialized atoms. These methods will also exploit indices and caches, and so the performance can be very good as we have demonstrated with the NRQL instantiation. Also the *cost-based* SUQL *query optimizer* is easily configurable (some methods must be overridden).

In order to decide entailment (as needed for query answering), inference algorithms which "work on substrates" must be called. In order to realize the integrated approach (and to address **P1 – P7**), our framework includes the MIDELORA[4] toolkit for DL system crafting. MIDELORA allows for the definition of specialized provers for certain tasks, working on specialized substrates. *Provers* are conceived as regions (or single points) in the three-dimensional MIDELORA **space**:

Definition 2 (MIDELORA Space)**.** The MIDELORA space is the cartesian product $\mathcal{S} \times \mathcal{L} \times \mathcal{T}$, where $\mathcal{S}$ is the set of *substrate classes*, $\mathcal{L}$ is the set of supported *(DL) languages*, and $\mathcal{T}$ is a set of *prover tasks.* $\triangle$

For example, $\mathcal{T}$ can contain the DL *standard inference problems* [8]: $\mathcal{T} = \{\mathsf{abox\_consistent?}, \mathsf{concept\_instances}, \ldots\}$. Again, substrates, languages and tasks are modeled as CLOS classes. A MIDELORA *prover* is a ternary multi-method with arguments $\langle S, L, T\rangle \in \mathcal{S} \times \mathcal{L} \times \mathcal{T}$. Polymorphism is exploited for all three arguments. Since inheritance is exploited for the definitions of the classes (elements) in the sets $\mathcal{S}$, $\mathcal{L}$, and $\mathcal{T}$, a single MIDELORA prover defined for a point (S, L, T) can cover a whole region in the MIDELORA space.

3.3 Benefits of the Framework

The problems **P1 – P7** are tackled as follows:

P1, "DL applicability problem". For some IS domains, there may be informational aspects which cannot be represented in a single representational framework (e.g., an ABox) or their representation is difficult or impossible with a *standard* DL ABox. Different substrate classes thus provide different extensional representation means. Substrates can also be *hybrid* and thus allow creation of layered representations: In a hybrid substrate, a DL ABox can be combined with some other arbitrary substrate, e.g., an SBox. Thus, the DL applicability problem can be defused pragmatically.

[3] However, this is not meant to reopen the "declarative vs. procedural" debate; instead, our framework shows that both approaches can and have to live together well, given that appropriate abstractions are provided which are "on the right level" for both perspectives.

[4] Michael's Description Logic Reasoner Architecture

P2, "Data and Expressivity Scalability Problem". DLs form a whole *family* of representation languages. In principle, DLs account for *expressivity scalability*. *Data scalability* can nowadays be achieved for simpler DLs, or RDF(S). In order to achieve data scalability, not only the knowledge level, but also the symbol level must also be considered. A *(persistent) database substrate* can be used if the extensional data is extensive (substrate graphs are then stored in an RDMS). Thus, the framework accounts for data scalability. It also accounts for expressivity scalability, since MIDELORA allows for the definition of language-specific provers. However, the services of standard DL systems such as RACER are also available to the framework.

P3, "Interoperability and Middleware Problem". The substrate data model can provide an abstraction layer on top of which the OBIS is built. This abstraction layer can, for example, shield the client code of the OBIS from details in the APIs of different DL systems (see also the *Design Patterns Adapter, Bridge and Facade* in [52]). A substrate can offer caching mechanisms, abstract from remote vs. local API procedure calls, etc. A substrate *can* thus also play the role of a *mediator or semantic middleware.* The remote componentware system need not even be a DL system, but can also be an RDF(S) triple store, an RDMS on which a graph view is established, etc.

Since substrates are CLOS classes utilizing inheritance which implement interfaces, additional services can easily be offered by means of substrate sub-classing. For example, a RACER *substrate* class will offer unique RACER services as methods in addition to the methods that are inherited from its *DL system substrate* superclass.

P4, "Missing Storage Layer Functionality Problem". Given that a substrate is not only a conceptual data model (an abstract data type on the knowledge level) but also implemented as a CLOS class, it is obvious that, by means of programming, the framework offers the flexibility to address and parameterize the storage layer. made persistent in a file or a MYSQL database.

P5, "Extensibility Problem". Extensibility and openness of the architecture is obviously realized, since object-orientation supports the well-known "Open-Closed Principle". However, reuse in frameworks has been identified as problematic in some cases, because inheritance-based reuse is "white box reuse" which thus requires knowledge about the internals of the class machinery. It is known that *domain specific languages (DSLs)* can resolve some of these problems [150].

P6, "Missing QL Problem". To address this problem, the SUQL engine is provided, which is as open, extensible and parameterizable as the rest of the framework. Decidability is guaranteed given that the required entailment relationship is decidable. From a theoretician's point of view, SUQL offers unions of grounded conjunctive queries (see Section 3.5).

P7, "Software-Abstraction Problem". We have argued that appropriate domain-specific software abstractions shall be provided in order to ensure maintainability and comprehensibility of a DL system and to avoid the "big ball of mud" (as understood in Software Engineering) syndrome in the life of a DL system.

Our approach is to define many small, comprehensible and specific provers for specific problems instead of just one big prover (implementing the core inference problem for a very expressive DL). The MIDELORA space provides the general structure for pinpointing provers. It provides a domain-specific software abstraction. The different provers are more comprehensible and concise than one big prover, since optimization techniques can be better localized (see Section 1.2.1). However, a big number of smaller provers can only be more maintainable and comprehensible if appropriate software abstractions are provided. MIDELORA offers *prover definition languages*, which can be understood as DSLs. Provers defined in these DSLs are almost as concise and comprehensible as the mathematical tableaux calculi used for DLs [74, 17].

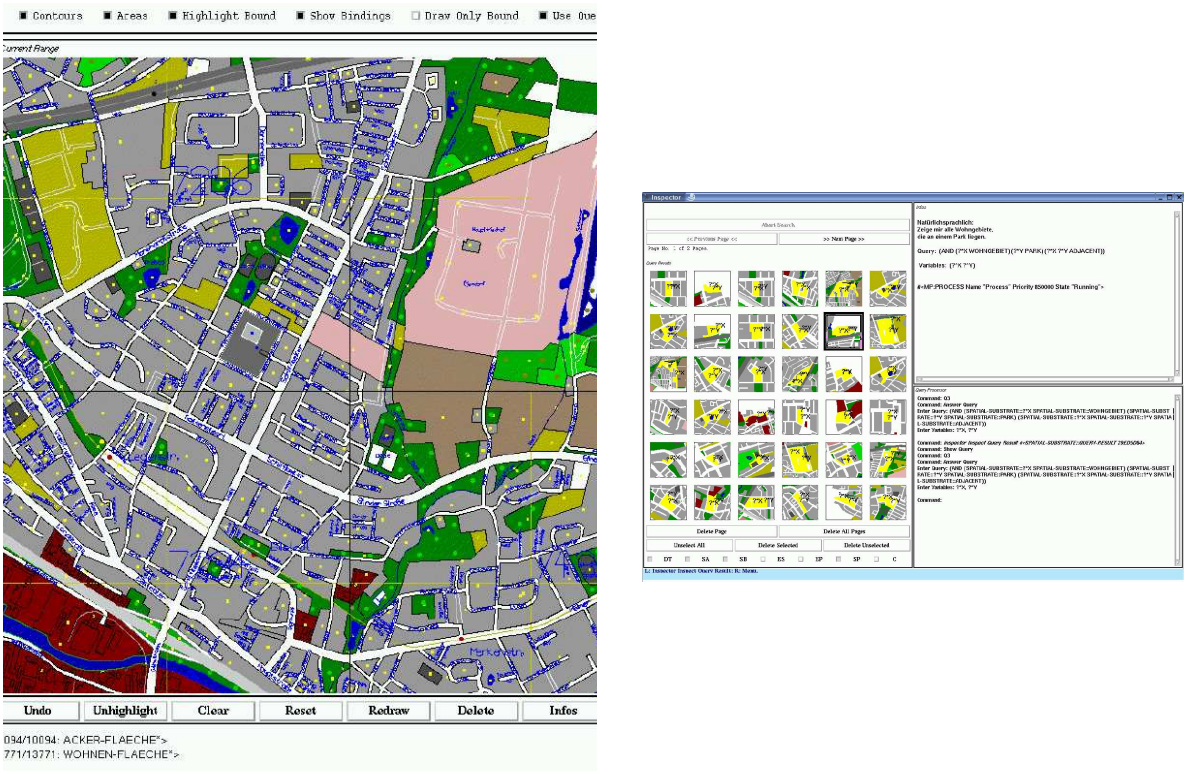

Figure 4: The MAP VIEWER and QUERY INSPECTOR of the DLMAPS System

3.4 Application Example DLMAPS: Ontology-Based Queries to City Maps

We now describe the digital city maps scenario. As mentioned, we are primarily using RACER as our standard DL component reasoner, but other setups are possible as well (some of these are described in the following).

3.4.1 The DISK Data

We are using digital vector maps of the city of Hamburg provided by the land surveying office ("Amt für Geoinformation und Vermessungswesen Hamburg"); these maps are called the DISK ("Digitale Stadtkarte"). Part of the DISK is visualized by the MAP VIEWER component of our system in Fig. 4. Each map object (also called *geographic feature*) is *thematically annotated*. The basic *thematic annotations (TAs)* have been established by the land surveying office itself. These TAs say something about the "theme" or semantics of the map objects. Simple concept names such as "green area", "meadow", "public park", "lake" are used. A few hundred TAs are used and documented in a so-called *thematic dictionary (TD)*, which is organized in so-called (thematic) *layers* (e.g., one layer for infrastructure, one for vegetation, etc.).

Sometimes, only highly specific TAs are available, such as "Cemetery for Non-Christians", and generalizing *common sense vocabulary*, e.g. "Cemetery", is missing. This is unfortunate, since it prevents the intuitive usage of common sense natural language vocabulary for query formulation, especially for non-casual users. We have repaired this defect by adding a background ontology (in the form of a TBox) providing generalizing TAs by means of taxonomic relationships.

On the other hand, *defined concepts* ("if and only if") can be added and exploited to *automatically enrich* the given basic annotations. Thus, we might define our own required TA "public park containing

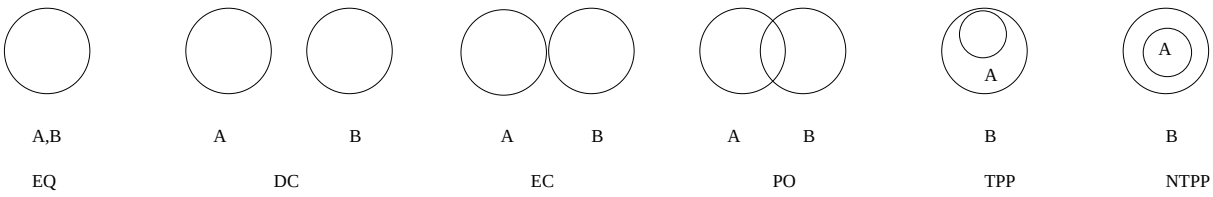

Figure 5: RCC8 base relations: EQ = *Eq*ual, DC = $D$is$c$onnected , EC = Externally Connected, PO = Partial Overlap, TPP = Tangential Proper Part, $NTPP$ = Non-Tangential Proper Part. All relations with the exception of TPP and $NTPP$ are symmetric; the inverse relations of TPP and $NTPP$ are called $TPPI$ and $NTPPI$.

a lake" as a "park which is public and contains a lake" with a TBox axiom such as

$$public\_park\_containing\_a\_lake \doteq park \sqcap public \sqcap \exists contains.lake$$
$$bird\_sanctuary\_park \doteq park \sqcap \forall contains.\neg building$$

and we might want to retrieve the instances of these concepts. This means that such instances must be recognized automatically, and this is what ontology-based query answering is all about. Obviously, inference is required to obtain these instances, since there are no *told* instances of $public\_park\_containing\_a\_lake$. For simple queries, simple *instance retrieval queries* might be sufficient. However, for reasons of expressivity and because we want to retrieve *constellations*[5] *of map objects,* a QL with variables is needed whose answer tuples can be visualized as in Fig. 4.

A definition such as $public\_park\_containing\_a\_lake$ refers to *thematic as well as to spatial aspects* of the map objects:

Thematic aspects: the name of the park, that the park is public, the amount of water contained in the lake, etc.

Spatial aspects: the *spatial attributes* such as the area of the park (or lake), the concrete shape, qualitative *spatial relationship* such as "contain", quantitative (metric) spatial relationships such as distance, etc.

We use the following terminology: a *thematic concept* refers only to thematic aspects, whereas a *spatial concept* refers solely to spatial aspects. A *spatio-thematic concept* refers to both. In the same sense we are using the terminology *thematic*, *spatial* and *spatio-thematic queries*.

Thus, there are different thematic and spatial aspects one would like to represent in the extensional component and subsequently query. Since the concrete geometry is given in the map, the spatial aspects of the map objects are in principle intrinsically represented and available. This mainly concerns the spatial relationships which are depicted in the map. However, spatial attributes such as the area or length of a map object can in principle also be derived (computed from the geometry), although this will not be very accurate. A function which exploits the map geometry to compute or verify a certain spatial aspect (for example, whether a certain qualitative relationship holds between two map objects) is called an *inspection method* in the following. This notion is defined as follows:

Definition 3 (Inspection Method)**.** Let S be an SBox, and P be a spatial FOPL formula without free variables (for example, an RCC ground atom such as $EC(a, b)$, where $a, b \in V$). An *inspection method* is a (geometric) algorithm which exploits the *geometry of S* to decide whether $S \models P$ holds. $\triangle$

It is obvious that *qualitative* spatial descriptions are of great importance. On the one hand, they are needed for the definitions of concepts in the TBox such as "public park containing a lake". On the other

[5]We use the term "constellation" to stress that a certain spatial arrangement of map objects is requested with a query.

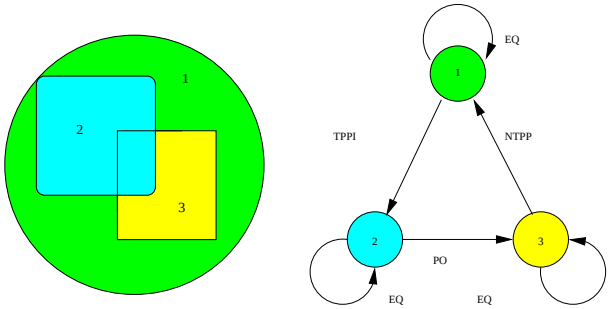

Figure 6: Geometric constellation and its RCC network (inverse edges omitted)

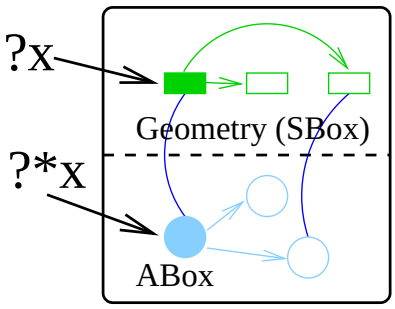

Figure 7: Hybrid substrate: variables are bound in parallel

hand, they are needed in the spatio-thematic QL ("retrieve all public parks containing a lake"). A popular and well-known set of qualitative spatial relationships is given by the RCC8 relations [126], see Fig. 5.

On the other hand, since the concrete geometry is given by means of the map, in principle, *no qualitative representation is needed* in the extensional component, since it can be reconstructed at query answering time by means of inspection methods. However, if we want to use a (standard-DL) ABox for the extensional component, then the spatial representation options are limited, and we must primarily resort to qualitative descriptions.

3.4.2 Representing and Querying the DISK

Representing qualitative spatial aspects and reasoning with this information is important in many application scenarios. Spatial representations are possible e.g.., with the already mentioned RCC8 calculus. Computational complexity of the RCC8 calculus is well studied and it is known that reasoning in the full RCC8 is NP-hard ([129]). Maximal tractable fragments of RCC8 were identified that includes all base relations but are satisfiable in polynomial time. The practical application of these fragments is still under investigation ([130]). In the domain of specialized logics, spatial representations can be done with expressive spatial *concrete domains (CDs)* [64, 102] or specialized DLs [154] or spatial modal logics [104]. However, many of these logics are either undecidable, or if they are decidable, no mature DL system supporting these non-standard DLs exists. In principle, MIDELORA allows for the definition of tableaux provers for such specialized languages. However, in this paper we focus on more pragmatic representations which incorporate RACER.

It is clear that the kind of representation we will devise for the DISK in the extensional component also determines what and how we can query. Without doubt, the thematic aspects of the DISK map objects can be represented satisfactorily with a standard DL. To solve the spatial representation problem of the DISK in the extensional component, we consecutively consider four different representation options and analyze their impacts.

3.4.3 Representation Option 1 – Simply Use an ABox

We can try to represent as many spatial aspects as possible in the ABox, given the DL supported by the exploited DL system, e.g. $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ in the case of RACER. Regarding the spatial relationships, we can only represent *qualitative* relationships. We can compute a so-called *RCC network* from the geometry of the map and represent this by means of *RCC role assertions* in the ABox, e.g. $(i, j) : TPPI$ etc. In Fig. 3.4.2(a) a "geometric scene" and its corresponding RCC8 network is depicted. Such a

network will always take the form of an edge-labeled complete graph[6], due to the *JEPD property* of the RCC base relations: The base relations are *j*ointly *e*xhaustive and *p*airwise *d*isjoint). Moreover, an RCC network derived from a geometric scene will always be *RCC consistent* (see Section 4.4).

Moreover, selected spatial attributes such as area and length can be represented in the ABox utilizing the concrete domain by means of concept assertions such as $i : \exists(has\_area).\ =_{12.345}$.

Since the represented spatial aspects are accessible to RACER, this supports spatio-thematic concept definitions in the TBox, for example

$$public\_park\_containing\_a\_lake \doteq park \sqcap public \sqcap \exists contains.lake$$

($\exists contains.lake$ is short for $(\exists TPPI.lake) \sqcup (\exists NTPPI.lake)$ for reasons of readability), the framework recognizes these qualitative spatial relationships and rewrites the query accordingly). Obviously, an individual i in the ABox can only be recognized as an instance of that concept if appropriate RCC role assertions are present as well.

In principle, the specific properties of qualitative spatial (RCC) relationships cannot be captured completely within $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ as roles (we will elaborate on this point below when we discuss qualitative spatial reasoning with the RCC substrate). This means that the computed taxonomy of the TBox will not correctly reflect the intended subsumption relationships. However, MIDELORA also supports $\mathcal{ALCI}_{\mathcal{RCC}}$ [154, 153]. Even though this DL is undecidable [104], the corresponding prover has successfully computed taxonomies of $\mathcal{ALCI}_{\mathcal{RCC}8}$ TBoxes. Moreover, the deduced *implied subsumption relationships* can be made syntactically explicit by means of additional TBox implication axioms, and this augmented TBox can be used instead of the original one in RACER.

Much more important in our scenario is the observation that *ontology-based query answering* can still be achieved in a way that correctly reflects the semantics of the spatial (RCC) relationships with RACER. Consider the *instance retrieval query* $public\_park\_containing\_a\_lake(?x)$ on the ABox

$$\mathcal{A} = \{i : park \sqcap public, k : lake, j : meadow, (i,j) : TPPI, (j,k) : NTPPI, \ldots\}$$

Since this ABox has been computed from the concrete geometry of the map, it must also contain $(i,k) : NTPPI$, because a RCC network which is computed from a spatial constellation that shows $(i,j) : TPPI$ and $(j,k) : NTPPI$ must necessarily also show $(i,k) : NTPPI$.

In order to retrieve the instances of $public\_park\_containing\_a\_lake$, we consider and check each individual separately. Let us consider i. Verifying whether i is an instance of $public\_park\_containing\_a\_lake$ is reduced to checking the unsatisfiability of $\mathcal{A} \cup \{(i,k) : NTPPI\} \cup \{i : \neg public\_park\_containing\_a\_lake\}$, or

$$\mathcal{A} \cup \{(i,k) : NTPPI\} \cup \{i : (\neg park \sqcup \neg public \sqcup ((\forall NTPPI.\neg lake) \sqcap (\forall TPPI.\neg lake)))\}$$

This ABox is unsatisfiable; thus, i is a $public\_park\_containing\_a\_lake$.

Regarding query concepts that contain or imply a *universal role or number restriction,* we can answer queries completely only if we turn on a "closed domain reasoning mode". We must *close the ABox w.r.t. the RCC role assertions* and *enable the Unique Name Assumption (UNA)*[7] in order to keep the semantics of the RCC roles. To close the ABox $\mathcal{A}$ w.r.t. the RCC role assertions, we count the number of *RCC role successors* of each individual for each RCC role: for $i \in \mathsf{individuals}(\mathcal{A})$ and the RCC role R, we determine the number of R-successors $n = |\{\, j \mid\ (i,j) : R \in \mathcal{A}\,\}|$ and add the so-called *number restrictions* $i : (\leq_n\ R) \sqcap (\geq_n\ R)$ to $\mathcal{A}$. This concept assertion is satisfied in an interpretation $\mathcal{I}$ iff $n = \{\, x \mid\ (i^{\mathcal{I}}, x) \in R^{\mathcal{I}}\,\}$; thus, i must have exactly n R successors in every model. In combination with the *Unique Name Assumption (UNA),* this turns on a closed domain reasoning on the individuals which are mentioned in the RCC role assertions and thus prevents the reasoner from the generation of "new anonymous RCC role successors" in order to satisfy an existential restriction such as $\exists NTPPI.lake$.

[6] Such a graph is called a K_n in graph theory.

[7] The UNA enforces that different individuals i, j are interpreted as different domain individuals in the interpretation: $i^{\mathcal{I}} \neq j^{\mathcal{I}}$.

In order to satisfy $\exists NTPPI.lake$, the prover must thus necessarily *reuse* one of the existing RCC role fillers from the ABox [157].

Let us demonstrate this technique using the query concept

$$bird\_sanctuary\_park \dot{=} park \sqcap \forall contains.\neg building.$$

Assuming that both $lake$ and $meadow$ imply $\neg building$, we can show that i is an instance of a $bird\_sanctuary$, since the ABox

$$\begin{aligned}\mathcal{A} \cup\ & \{(i,k) : NTPPI\} \cup \\ & \{i : (\leq_1\ TPPI) \sqcap (\geq_1\ TPPI), i : (\leq_1\ NTPPI) \sqcap (\geq_1\ NTPPI), \ldots\} \cup \\ & \{i : (\neg park \sqcup ((\exists TPPI.building) \sqcap (\exists NTPPI.building)))\}\end{aligned}$$

is again unsatisfiable, because the alternative $i : \neg park$ immediately produces an inconsistency. Thus, the alternative $i : (\exists TPPI.building) \sqcap (\exists NTPPI.building)$ is considered. Due to $i : (\leq_1\ TPPI) \sqcap (\geq_1\ TPPI)$, only j can be used to satisfy $\exists TPPI.building$, and only k to satisfy $\exists NTPPI.building$. Since $j : meadow$ and thus $j : \neg building$, $k : lake$ and thus $k : \neg building$, the ABox must be unsatisfiable.

Thus, we have argued that spatio-thematic ontology-based query answering can be done on such an ABox representation of the DISK, and that this is to some extent – using some logical encoding tricks – possible *even with simple instance retrieval queries.*

Using an Expressive ABox Query Language We now demonstrate that the RACER ABox query language nRQL [68, 69] offers valuable additional query formulation facilities in this scenario. For now, we are using grounded conjunctive queries in mathematical (Horn-logic) syntax and assume that the reader has an *intuitive understanding (in addition to our explanations).* The semantics of SuQL (and nRQL) will be defined formally in Section 5 (Section 5.1). We demonstrate that nRQL's *negation as failure (NAF negation)* enables a great deal of differentiation possibilities for query formulation. For example, we can *query for living areas adjacent to parks which contain a lake ...*

(1) ... *which are* provably not *adjacent to industrial areas.* Thus, all adjacent areas are provably not industrial areas (note that $adjacent$ is recognized as synonym for EC):

$$\begin{aligned}& ans(?living\_area, ?park, ?lake) \leftarrow \\ & \quad living\_area(?living\_area), park(?park), lake(?lake), \\ & \quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\ & \quad (\forall adjacent.\neg industrial\_area)(?living\_area)\end{aligned}$$

(2) ... *for which there are no adjacent industrial areas known* (NAF negation):

$$\begin{aligned}& ans(?living\_area, ?park, ?lake) \leftarrow \\ & \quad living\_area(?living\_area), park(?park), lake(?lake), \\ & \quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\ & \quad \backslash(\exists adjacent.industrial\_area(?living\_area))\end{aligned}$$

Slightly simplified, the subquery $\backslash(\exists adjacent.industrial\_area(?living\_area))$ first retrieves the instances of the concept $\exists adjacent.industrial\_area$, and then simply builds the complement set (this explains the use of "\"). Thus, a candidate binding for $?living\_area$ must be in that complement set. Please note that the instances of $\forall adjacent.\neg industrial\_area$ form a subset of this set.

(3) ... *for which there are no known adjacent industrial areas known:*

$$ans(?living\_area, ?park, ?lake) \leftarrow$$
$$living\_area(?living\_area), park(?park), lake(?lake),$$
$$contains(?park, ?lake), adjacent(?living\_area, ?park),$$
$$\backslash(\pi(?living\_area)\ \ adjacent(?living\_area, ?i), industrial\_area(?i))$$

The subquery $\backslash(\pi(?living\_area)\ \ adjacent(?living\_area, ?i), ind.\_area(?i))$ returns the complement set of the answer to the query
$ans(?living\_area) \leftarrow adjacent(?living\_area, ?i), ind.\_area(?i))$.[8]

So, an instance is in $\backslash(\pi(?living\_area)\ \ adjacent(?living\_area, ?i),\ ind.\_area(?i)$ iff for $?living\_area$ there is no known adjacent industrial area present. However, in principle $?living\_area$ might have an *unknown* adjacent industrial area (in case there is no corresponding ABox individual) – thus, this query returns a *superset* of $\backslash(\exists adjacent.industrial\_area(?living\_area))$, and the query is therefore *more general* than (2).

Drawbacks of the ABox Representation Even though ontology-based query answering is sort of possible using the just discussed ABox representation, it nevertheless has the *following drawbacks:*

1. The size of the generated ABoxes is significant. Since the RCC network is explicitly encoded in the ABox, the number of required role assertions is quadratic in the number of map objects, $|V|^2$ (several million role membership assertions for the DISK).

2. Most spatial aspects cannot be handled that way. For example, distance relations are very important for map queries. It is thus not possible to retrieve all subway stations within a distance of 100 meters from a certain point.

3. Query processing will not be efficient for queries which mention spatial aspects, since spatial index structures are missing.

4. In the DLMAPS system, the geometric representation of the map is needed anyway, at least for presentation purposes. Thus, from a non-logical point of view, the ABox cannot be the only representation used in the extensional component of such a system. Thus, it seems plausible to exploit this geometric representation for query answering as well.

5. Most importantly, we have demonstrated that this kind of ontology-based query answering works only if the domain is "RCC closed". However, DL systems are *not really good at closed domain reasoning,* since the *Open Domain Assumption (ODA)* is made in DLs. This will be illustrated in Section 4.3.

   In contrast, since the geometry of the map is completely specified, there is neither unknown nor underspecified spatial information. This motivates the classification of such a map as spatial *data*. We thus switch to a hybrid representation incorporating an SBox.

3.4.4 Representation Option 2 – Use a Map Substrate:

Due to the problems with spatio-thematic concepts and since closed domain reasoning is all that we can achieve here anyway, it seems more appropriate to represent the spatial aspects *primarily* in the SBox (a kind of "spatial database"), and *associate* an ABox with that SBox. We have already mentioned that the geometry of the map must be represented in the extensional component anyway (at least for presentation

[8] Please note that π is called the *body projection operator,* see Section 5.

purposes). If we say that the spatial aspects are *primarily* represented in the SBox, then this does *not* necessarily exclude the (additional) representation possibilities of dedicated spatial aspects in the ABox as just discussed.

The resulting hybrid $(SBox, ABox)$ representation is illustrated in Fig. 3.4.2(b); we call it a *map substrate*. The figure illustrates that some ABox individuals have corresponding instances in the SBox, and vice versa. A partial and injective mapping function "$*$" which maps nodes in the SBox to nodes in the ABox (and vice versa, $*^{-1}$) is used. Thus, we first define a *hybrid substrate* and a *map substrate* as follows:

Definition 4. A *hybrid substrate* is a triple $(S_1, S_2, *)$, with S_i, $i \in \{1, 2\}$ being substrates $(V_i, E_i, L_{V_i}, L_{E_i})$ using $\mathcal{L}_{\mathcal{V}i}$ and $\mathcal{L}_{\mathcal{E}i}$, $*$ being a partial and injective function $* : V_1 \mapsto V_2$. A *map substrate* is a hybrid substrate $(S_1, S_2, *)$, where S_1 is an SBox, and S_2 is an ABox. △

If the spatial aspects of the DISK are now *primarily* kept in the SBox, then they are no longer necessarily available for ABox reasoning and retrieval. Thus, NRQL (or instance retrieval) queries are no longer sufficient to address these spatial aspects – we will thus extend NRQL to become a hybrid spatio-thematic QL, also offering *spatial query atoms* to query the SBox: SNRQL.

The SNRQL query answering engine will combine the retrieved results from the SBox with results from the ABox. The thematic part of such a SNRQL query is given by a plain NRQL query, and the spatial part utilizes spatial query atoms which are evaluated on the SBox by means of inspection methods. The SBox provides a spatial index, supporting the efficient evaluation of inspection methods by means of spatial selection operations. Computed spatial aspects can also be materialized in order to avoid repeated re-computation (e.g., RCC relations can be materialized as edges).

Given a hybrid substrate, a *hybrid query* now contains two kinds of query atoms: Those for S_1, and those for S_2. In order to distinguish atoms meant for S_1 from atoms meant for S_2, we simply prefix variables in query atoms for S_2 with a "?$*$" instead of "?"; the same applies to individuals. Intuitively, the *bindings* which will be established for variables must also reflect the $*$-function: If $?x$ is bound to $i \in V_1$, then $? * x$ will automatically be bound to $*(i) \in V_2$ (if defined), and vice versa (w.r.t. $*^{-1}$). Such a binding is called $*$-*consistent*. We will only consider such $*$-consistent bindings. The notion of a $*$-consistent binding is also depicted in Fig. 3.4.2.

Assume we are using a map substrate for the DISK representation. Let us consider the example query given in Section 3.4.3 again. Since the RCC network is now no longer represented in the ABox, the SBox must be queried for spatial relationships. Queries (1) and (2) from Section 3.4.3 thus no longer work.

However, query (3) has a "SNRQL equivalent" which looks as follows. Note that NRQL query atoms now use $*$-prefixed variables, since the ABox is S_2, and the SBox is S_1:

$$
\begin{array}{l}
ans(?living\_area, ?park, ?lake) \leftarrow \\
\quad living\_area(? * living\_area), park(? * park), \\
\quad contains(?park, ?lake), adjacent(?living\_area, ?park), \\
\quad \setminus\ (\ \pi(?living\_area)\ (\ adjacent(?living\_area, ?industrial\_area), \\
\qquad\qquad\qquad\qquad industrial\_area(? * industrial\_area)))
\end{array}
$$

Thus, we not only gain, but also lose something here (queries (1) and (2) cannot be expressed). This is an important insight. On the positive side, we are now able to define and evaluate spatial predicates which are richer than RCC predicates, since the geometry of the map is represented. We can thus design dedicated spatial query atoms. These spatial atoms (e.g., distance query atoms) are discussed in Section 5.2.

3.4.5 Representation Option 3 – Use a Spatial ABox

Using the MIDELORA toolkit, we can define provers working on specialized substrate classes. We already mentioned in Section 3.1 that MIDELORA offers so-called *spatial ABoxes*. There is then no longer a need for a hybrid map representation, since ABox individuals are also instances of spatial datatypes (like SBox nodes). From the point of view of a *standard* DL prover in MIDELORA, the spatial aspects of these nodes are invisible. However, dedicated "spatial" MIDELORA provers or query answering procedures (implementations of spatial query atoms) can be defined which exploit the spatial aspects of the nodes.

With a spatial ABox, the RCC role assertions need not be precomputed and added as assertions at all. They can be computed by means of inspection methods and materialized on the fly *if needed* during the tableau proof. Thus, there is no need to explicitly store an $|V|^2$ number of RCC role assertions in the ABox, as they are "intrinsically represented". However, this requires a dedicated prover which can be defined in MIDELORA.

In Section 4.1 we have closed the ABox w.r.t. the RCC role assertions. As explained, the $i : (\leq \ R\ n) \sqcap (\geq \ R\ n)$ number assertions force the tableaux prover to reuse existing ABox individuals when existential (successor generating) concepts are expanded which reference an RCC role. This forces RACER into a *closed domain reasoning mode*; however, this is a *two-step process in the* $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ *tableau calculus*. First, a fresh node satisfying the existential concept is created. Then, later on in the tableaux expansion process, it is found that this fresh node contradicts the $(\leq \ R\ n)$ assertion. Thus, the so-called *merge rule* identifies and merges the superfluous successors with an already existing R successor (mentioned in a role assertion). However, this is a *highly non-deterministic process.* Thus we stated in Section 4.1.2 (5) that DL reasoners are not very good at closed domain reasoning.

It is obvious that this behavior of the tableaux prover could also be achieved in a more direct way if the generating rules were modified in such a way that first the *reuse of an existing successor* is tried before a fresh successor is generated for an RCC role. (However, the generating rules become non-deterministic with that modification). The tableaux rules of MIDELORA can be parameterized to work in such a way.

3.4.6 Representation Option 4 – Use an ABox + RCC Substrate

Finally, we can discuss a fourth option. The primary motivation for this option is to make some spatial functionality available to other users of the RACER system. Thus, in order to offer a comparable spatio-thematic query answering functionality to other users of the RACER system without having to add the whole SBox functionality to RACER (spatial datatypes), we devise yet another kind of substrate, the *RCC substrate*, which captures the semantics of the RCC relations by exploiting techniques from *qualitative spatial reasoning.* Unlike the $\models$ relation for the SBox, which only exploits spatial model checking by means of inspection methods, spatial inference is thus required here. The RCC substrate is, on the one hand, more expressive then the SBox, since also vague or unknown RCC relations can be expressed. On the other hand, the geometry of the map cannot be preserved (as in Option 1).

Users of RACER can associate an ABox $\mathcal{A}$ with an RCC substrate $\mathcal{RCC}$ by means of a hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ and query this hybrid substrate with NRQL + *RCC query atoms* (see Section 5.3). Unlike for the map substrate, the ABox is the primary substrate S_1, since the RCC substrate is an "add on" from the perspective of the RACER user. Let us describe the RCC substrate:

Definition 5. Let $\mathcal{R} =_{def} \{EQ, DC, EC, PO, TPP, TPPI, NTPP, NTPPI\}$ be the set of RCC8 base relations. An *RCC substrate* $\mathcal{RCC}$ is a substrate such that V is a set of RCC nodes with $\mathcal{L}_{\mathcal{V}} = \emptyset$, and $\mathcal{L}_{\mathcal{E}} = 2^{\mathcal{R}}$. $\triangle$

The RCC base relations have already been discussed. An edge label represents a *disjunction of RCC base relations*, representing coarser or even unknown knowledge regarding the spatial relation (where the set is not a singleton). Disjunctions of base relations are thus RCC relations as well. The *properties* of the RCC relations are captured by the so-called JEPD property (see Page 36) as well as the so-called *RCC composition table*. This table is used for solving the following basic inference problem: **Given: RCC relations** $R(a,b)$ **and** $S(b,c)$**. Question: Which relation** T **holds between** a **and** c**?** The table thus lists, at column for base relation R and row for base relation S, the RCC relation T. In general, T will not be a base relation, but a set denoting a disjunctive RCC relation: $\{T_1, \ldots T_n\}$. The RCC table is given as a set $\mathcal{RCC}_\mathcal{T}$ of sentences of the form $\{R \circ S = \{T_1, \ldots, T_n\}, \ldots\}$.

An RCC substrate $\mathcal{RCC}$ *containing only base relations* can be viewed as a set of FOPL ground atoms. Such a RCC network is said to be *relationally consistent* iff $\mathcal{RCC}'$ is satisfiable:

$$\begin{aligned}\mathcal{RCC}' = \ & \mathcal{RCC} \cup \{\forall x : EQ(x,x)\} \cup \\ & \{\forall x,y,z : R(x,y) \wedge S(y,z) \rightarrow T_1(x,z) \vee \cdots \vee T_n(x,z) \mid \\ & \qquad\qquad R \circ S = \{T_1, \ldots, T_n\} \in \mathcal{RCC}_\mathcal{T}\} \cup \\ & \{\forall x,y : \bigvee_{R \in \mathcal{R}} R(x,y)\} \cup \{\forall x,y : \bigvee_{R,S \in \mathcal{R}, R \neq S} R(x,y) \wedge \neg S(x,y)\}\end{aligned}$$

For example, the network $\mathcal{RCC} = \{NTPPI(a,b), DC(b,c), PO(a,c)\}$ is inconsistent, because if a is contained in b (atom $NTPPI(a,b)$), and b is disconnected from c (atom $DC(b,c)$), then a must be disconnected from c as well. The *RCC8 composition table* contains the axiom $NTPPI \circ DC = \{DC\}$. Thus, $\mathcal{RCC}' \models DC(a,c)$, which contradicts $PO(a,c)$, due to the JEPD property.

Let us briefly define some more notions. *Entailment of RCC relations* or RCC ground query atoms can be *reduced to inconsistency checking* as follows: $\mathcal{RCC}' \models R(a,b)$ iff $\mathcal{RCC}' \cup \{(\mathcal{R} \setminus R)(a,b)\}$ is unsatisfiable. *A (general) RCC network is relationally consistent* iff at least one of its *configurations* is relationally consistent. A *configuration of an RCC network* is obtained by choosing (and adding) one disjunct / base relation out of every non-base relation in that network (thus, a configuration contains only base relations).

For example, consider $\mathcal{RCC} = \{NTPP(a,b), DC(b,c)\}$. We have $\mathcal{RCC}' \models DC(a,c)$, since $\mathcal{RCC}' \cup \{EQ, EC, PO, TPP, TPPI, NTPP, NTPPI\}(a,c)$ is not relationally consistent, because none of its *configurations* $\mathcal{RCC}' \cup \{EQ(a,c)\} \ldots \mathcal{RCC}' \cup \{NTPPI(a,c)\}$ is relationally consistent.

Since the RCC substrate defines a notion of logical entailment, the semantics of the RCC relations will be correctly captured for query answering. Consider the hybrid substrate $(\mathcal{A}, \mathcal{RCC}, *)$ with

$$\mathcal{A} = \{hamburg : german\_city, paris : french\_city, fr : country, ger : country\} \quad ,$$
$$\mathcal{RCC} = \{NTPP(*hamburg, *ger), EC(*ger, *fr), NTPP(*paris, *fr)\}$$

and with the obvious mapping $*(x) = *x$ for $x \in \{hamburg, paris, fr, ger\}$. Then, the query

$$ans(?city1, ?city2) \leftarrow city(?city1), city(?city2), DC(? * city1, ? * city2)$$

correctly returns $?city1 = hamburg, ?city2 = paris$, and vice versa, even though $DC(*paris, *hamburg)$ is not present in $\mathcal{RCC}$.

3.5 SUQL – The Substrate Query Language Framework

In the following we describe the core design principles underlying the generic substrate query language SUQL, its instantiations (NRQL, SNRQL), as well as the features and core optimizations found in the query answering engine.

Some ideas of the SUQL framework have already been outlined, and additionally some examples for queries using abstract Horn-logic syntax have been given. In the following, we will use the concrete syntax of the query language framework in order to make it less abstract.[9] The query

$$ans(?x, ?y) \leftarrow woman(?x), has\_child(?x, ?y)$$

[9] The prefix Lisp syntax is as readable and as formal as the mathematical syntax.

takes the following form in concrete syntax:

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

The expression `(?x ?y)` is called the *head,* and `(and (?x woman) (?x ?y has-child))` the *body* of the query. SUQL offers *substrate-specific* unary and binary *query atoms* (whose concrete syntax may be defined accordingly), from which *complex queries* can be constructed using the (generic) body constructors `and, or, neg` and `project-to`; `neg` corresponds to "\", and `project-to` to "π".

If we assume that `(?x woman)` is a *concept query atom*, – a specialized unary query atom for substrates of class *ABox* –, and `(?x ?y has-child)` is a *role query atom* – a specialized binary query atom for for substrates of class *ABox* –, then, if posed to a substrate of type ABox, the query returns all mother-child pairs from that ABox.

SUQL has the following peculiarities which we want to discuss briefly before syntax and semantics is specified:

Variables and individuals can be used in query atoms. Both variables and individuals are called *objects*. The variables range over V, the nodes of the substrate. Thus, SUQL offers only so-called *distinguished* or *must-bind variables* [84]. Variables are bound to nodes which *satisfy* the query – a variable binding satisfies a query iff the ground query – that is obtained from replacing all variables with their bindings – is logically entailed by the substrate. For example, the atom $P(x)$ is satisfied in substrate S if $x = i$, $i \in V$ and $S \models P(x)_{x \leftarrow i}$. Thus, a variable is only bound to a substrate node iff it can be proven that this binding holds in *all* models of the substrate.

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

Negation as Failure (NAF) Operator The `neg` operator implements a *Negation as Failure Semantics (NAF)*. For example, `(neg (?x woman))` returns all substrate nodes for which it *cannot be proven* that they are instances of `woman`. Thus, `(neg (?x woman))` returns the complement set of `(?x woman)` (w.r.t. V, the set of all substrate nodes). If a binary query atom is NAF negated, e.g. `(neg (?x ?y has-child))`, then the complement is two-dimensional. Thus, all pairs of individuals are returned which are not in the `has-child` relation.

Let us define the *extension* of a unary (binary) query atom $P(?x)$ ($Q(?x, ?y)$) as the query answer of the query $ans(?x) \leftarrow P(?x)$ (resp. $ans(?x, ?y) \leftarrow Q(?x, ?y)$, and denote that extension as $P(?x)^{\mathcal{E}}$ (resp. $Q(?x, ?y)^{\mathcal{E}}$). It is obvious that the following equalities must hold, for any substrate S with nodes V:

$$\begin{aligned} V &= P(?x)^{\mathcal{E}} \cup (\backslash P(?x))^{\mathcal{E}} \\ V \times V = V^2 &= Q(?x, ?y)^{\mathcal{E}} \cup (\backslash Q(?x, ?y)).^{\mathcal{E}} \end{aligned}$$

Let us consider the ABox query language case again. We would like to stress that `(?x (not woman))` has a different semantics from `(neg (?x woman))`, since the former returns the individuals for which the DL system *can prove* that they are *not instances* of woman, whereas the latter returns all instances for which the DL system *cannot prove* that they are *instances* of woman. Also note that `neg` and `not` are equivalent on substrates which employ the CWA (e.g., the SBox).

Different Notions of Equality are Available Equality atoms can either use syntactic or semantic equality predicates: "$=_{syn}$" or "$=_{sem}$"; these notions coincide if the UNA is used.[10]

[10]The predicate $=_{sem}$ is the standard equality predicate in FOPL with equality.

The Body Projection Operator (`project-to`) This operator is required in order to reduce the "dimensionality" of the extension of a subbody in a query body before the complement set is computed with `neg`. It allows to "fold in" subbodies for which dedicated horn rules would have to be written otherwise. For example, in order to retrieve those individuals which do *not* have a *known* child, we have to use `(neg (project-to (?x) (?x ?y has-child)))`, since the extension of `(neg (?x ?y has-child))` is a *two-dimensional set*.

3.5.1 Syntax and Semantics

We only specify syntax and semantics for non-hybrid queries. The extension to hybrid queries is straightforward, but does not really add to this paper.

Definition 6 (Syntax of SUQL)**.** The *head* and *body* of a SUQL query, `(retrieve` *head body*`)`, are defined by the following grammar ({a|b} means a or b):

$$\begin{array}{rcl} \textit{head} & := & \texttt{(}\textit{object}^*\texttt{)} \\ \textit{object} & := & \textit{variable} \mid \textit{individual} \\ \textit{variable} & := & \text{a symbol beginning with } \texttt{?} \\ \textit{individual} & := & \text{a symbol} \\ \textit{body} & := & \textit{atom} \mid \texttt{(}\{\texttt{and}\mid\texttt{union}\}\ \textit{body}^*\texttt{)} \mid \texttt{(neg}\ \textit{body}\texttt{)} \mid \\ & & \texttt{(project-to (}\textit{object}^*\texttt{)}\ \textit{body}\texttt{)} \\ \textit{atom} & := & \textit{unary\_atom} \mid \textit{binary\_atom} \mid \textit{equality\_atom} \\ \textit{unary\_atom} & := & \texttt{(}\textit{object}\ \textit{unary\_atom\_predicate}\texttt{)} \\ \textit{binary\_atom} & := & \texttt{(}\textit{object}\ \textit{object}\ \textit{binary\_atom\_predicate}\texttt{)} \\ \textit{equality\_atom} & := & \texttt{(}\textit{object}\ \textit{object}\ \{=_{syn} \mid =_{sem}\}\texttt{)} \end{array}$$

The predicates $unary\_atom\_predicate$ and $binary\_atom\_predicate$ are conceived as FOPL formulas with one (resp. two) free variables x and y; however, the concrete syntax may offer a variable-free syntax for them.

The function $\mathsf{obs}(q)$ returns the *objects (individuals and variables)* referenced in q and is defined inductively as follows: $\mathsf{obs}(unary\_atom) =_{def} \{x_1\}$ if $unary\_atom = \texttt{(}x_1\ unary\_atom\_predicate\texttt{)}$, $\mathsf{obs}(binary\_atom) =_{def} \{x_1, x_2\}$ if $binary\_atom = \texttt{(}x_1\ x_2\ Q\texttt{)}$ with $Q \in \{binary\_atom\_predicate, =_{syn}, =_{sem}\}$, $\mathsf{obs}(\texttt{(\{ and | union | neg \}}\ q_1 \ldots q_m\texttt{)}) =_{def} \bigcup_{1\le i\le m} \mathsf{obs}(q_i)$, but $\mathsf{obs}(\texttt{(project-to (}x_1 \ldots x_m\texttt{) ...)}) =_{def} \{x_1 \ldots x_m\}$. Thus, obs "stops at projections". △

Before we can define the semantics we need some auxiliary operations. 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 1s, and $\mathcal{B}$ is some set ("the base"), and $\mathcal{T}$ 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

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

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 1s 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 $\mathcal{B}$.

Definition 7 (Semantics of SUQL)**.** Let $S = (V, E, L_V, L_E, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$ be a substrate, and q be a *body*.

The semantics of a query is given by the set of tuples it returns if posed to a substrate S. This set of answer tuples is called the extension of q and denoted by $q^\mathcal{E}$.

First we add equality atoms for query atoms which reference individuals. The query body q is thus first rewritten. We define $\Theta(q)$ for $atom$ with $\mathsf{obs}(atom) \cap V = \{v_1, \ldots, v_n\}$, $n \in \{1,2\}$ as

$$\Theta(atom) =_{def} (\texttt{and}\ atom\ (x_{v_1}\ v_1\ =)\ \ldots\ (x_{v_n}\ v_n\ =)),$$

(please note that $= \in \{=_{syn}, =_{sem}\}$, as previously discussed, and that x_{v_i} is the representative variable for v_i) and extend the definition of Θ in the obvious (inductive) way to complex query bodies as well. Moreover, Θ replaces all occurrences of individuals in the projection list of `project-to` and in the query $head$ with their representative variables.

Let $q' = \Theta(q)$ be the rewritten query. So we simply declare $q^\mathcal{E} =_{def} q'^\mathcal{E}$. Let us specify $q'^\mathcal{E}$. Let $\langle x_{1,q'}, \ldots, x_{n,q'} \rangle$ be some fixed enumeration of $\mathsf{obs}(q')$ (so $n = |\mathsf{obs}(q')|$).

We define $\cdot^\mathcal{E}$ inductively. We start with the query atoms:

$$\begin{aligned}
(x_{i,q'}\ P)^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i\}}}(\{ < v > \mid v \in V, S \models P_{x \leftarrow v} \}\ ) \\
(x_{i,q'}\ x_{j,q'}\ Q)^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, S \models Q_{x \leftarrow u, y \leftarrow v} \}\ )
\end{aligned}$$

(please note that due to Θ, all unary and binary query atoms which are not equality atoms now have one and two variables correspondingly). The semantics of the equality predicates is fixed as follows: $S \models i =_{syn} i$ and $S \not\models i =_{syn} j$, and $S \models i =_{sem} j$ iff for all models $\mathcal{I}$ of S ($\mathcal{I} \models S$): $i^\mathcal{I} = j^\mathcal{I}$. Thus we define:

$$\begin{aligned}
(x_{i,q'}\ x_{j,q'}\ =_{syn})^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, \\
&\qquad \text{if } x_{i,q'} \in V, \text{then } u = x_{i,q'}, \text{if } x_{j,q'} \in V, \text{then } v = x_{j,q'} \}) \\
(x_{i,q'}\ x_{j,q'}\ =_{sem})^\mathcal{E} &=_{def} \chi_{V,\vec{\mathbf{1}}_{n,\{i,j\}}}(\{ < u,v > \mid u,v \in V, S \models u =_{sem} v, \\
&\qquad \text{if } x_{i,q'} \in V, \text{then } u = x_{i,q'}, \text{if } x_{j,q'} \in V, \text{then } v = x_{j,q'} \}).
\end{aligned}$$

We extend the definition of $\cdot^\mathcal{E}$ inductively for complex (sub)bodies in q':

$$\begin{aligned}
(\texttt{and}\ \ q'_1 \ldots q'_i)^\mathcal{E} &=_{def} \textstyle\bigcap_{1 \le j \le i} q'^\mathcal{E}_j \\
(\texttt{union}\ \ q'_1 \ldots q'_i)^\mathcal{E} &=_{def} \textstyle\bigcup_{1 \le j \le i} q'^\mathcal{E}_j \\
(\texttt{neg}\ \ q'_1)^\mathcal{E} &=_{def} V^n \setminus q'^\mathcal{E}_1 \\
(\texttt{project-to}\ \ (x_{i_1,q'} \ldots x_{i_k,q'})\ \ q'_1)^\mathcal{E} &=_{def} \pi_{\langle i_1,\ldots,i_k \rangle}(q'^\mathcal{E}_1)
\end{aligned}$$

To get the final answer of a query, the *head* has to be considered, for a final projection. Thus, the result of `(retrieve` $head$ q`)` is simply given as

$$(\texttt{retrieve}\ head\ q)^\mathcal{E} =_{def} (\texttt{project-to}\ \Theta(head)\ \Theta(q))\ .^\mathcal{E}$$

△

3.5.2 The nRQL Instantiation of the SUQL

In addition to the basic retrieval inference service, expressive query languages are required in practical applications. Well-established is the class of conjunctive queries. A *conjunctive query* consists of a *head* and a *body*. The head lists variables for which the user would like to compute bindings. The body consists of query atoms (see below) in which all variables from the head must be mentioned. If the body contains additional variables, they are seen as existentially quantified. A query answer is a set

of tuples representing bindings for variables mentioned in the head. A query is a structure of the form $ans(X_1, \ldots, X_n) \leftarrow atom_1, \ldots, atom_m$.

Query atoms can be *concept* query atoms ($C(X)$), *role* query atoms ($R(X, Y)$), *same-as* query atoms ($X = Y$) as well as so-called *concrete domain* query atoms. The latter are introduced to provide support for querying the concrete domain part of a knowledge base. Complex queries are built from query atoms using boolean constructs for conjunction (indicated with comma) or union ($\vee$).

In *standard* conjunctive queries, variables (in the head and in query atoms in the body) are bound to (possibly anonymous) domain objects. A system supporting (unions of) standard conjunctive queries is QuOnto. In so-called *grounded* conjunctive queries, $C(X)$, $R(X, Y)$ or $X = Y$ are true if, given some bindings α for mapping from variables to *individuals mentioned in the Abox* $\mathcal{A}$, it holds that $(\mathcal{T}, \mathcal{A}) \models \alpha(X) : C$, $(\mathcal{T}, \mathcal{A}) \models (\alpha(X), \alpha(Y)) : R$, or $(\mathcal{T}, \mathcal{A}) \models \alpha(X) = \alpha(Y)$, respectively. In grounded conjunctive queries the standard semantics can be obtained for so-called tree-shaped queries by using corresponding existential restrictions in query atoms. Due to space restrictions, we cannot discuss the details here. In the following, we consider only grounded conjunctive queries. RACER's Abox query language is described in [155]. The language is called nRQL (pronounce: "niracle" and hear it as "miracle").

nRQL [68, 69] is a specialized SuQL. It offers dedicated query atoms for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, e.g. atoms addressing the concrete domain part of an ABox. The nRQL atoms are: *concept query atoms*, e.g. `(?x (some has-child human))`; *role query atoms,* e.g. `(?x ?y has-child)`, and (binary) *constraint query atoms*. All atoms have been discussed already, with the exception of constraint query atoms. The following query uses all kinds of nRQL atoms:

```
(retrieve (?x)
      (and (?x (and woman (min age 40))) (?x ?y has-child)
           (?y ?y (constraint (has-father age) (has-mother age)
                              (<= (+ age-2 8) age-1)))))
```

This query returns thus instances of the concept `women` which are older than 40 and which have children whose fathers are at least 8 years older than their mothers. Note that `(has-father age)` denotes a role chain ended by a so-called concrete domain attribute, a kind of "path expression": starting from the individual bound to `?y` (the child), 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 check whether the predicate `(<= (+ age-2 8) age-1)` holds for them; `age-2` refers to `(has-mother age)`, and `age-1` refers to `(has-father age)`.[11] However, these "values" are in fact variables in a concrete domain constraint network (which can be left unspecified, i.e., no syntactically specified so-called *told value* must exist).

Also more general *role terms* are admissible in role and constraint query atoms; a role term is an element in the set of role names closed under the operators $\{\texttt{not}, \texttt{inv}\}$. Thus, nRQL offers not only NAF negated roles, but also *classical negated roles*, which are not provided by $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$.

Given the generic semantics definition, it should be clear how the semantics of the dedicated nRQL atoms can be defined. Basically, we just need to define $S \models P_{x \leftarrow v}$ as well as $S \models Q_{x \leftarrow u, y \leftarrow v}$; note that S is now an ABox $\mathcal{A}$. However, this is easy using the standard translation Φ of DL into FOPL [8]; e.g., for a concept query atom predicate $P = C$ this boils down to ordinary instance checking or an instance retrieval query: $\mathcal{A} \models \Phi(C)_{x \leftarrow i}$ iff $\mathcal{A} \models i : C$ iff $\mathcal{A} \cup \{i : \neg C\}$ is unsatisfiable (basically, just one of the RACER API functions concept_instances or individual_instance? need to be called), and for positive roles R in role atoms we get $\mathcal{A} \models \Phi(R)_{x \leftarrow i, y \leftarrow j}$ iff $\mathcal{A} \models (i, j) : R$ iff $\mathcal{A} \cup \{i : \forall R.M, j : \neg M\}$

[11] Note that the suffixes `-1, -2` have been added to the `age` attribute in order to differentiate the two values (the mechanism is not needed where the two chains end in different attributes).

is unsatisfiable, for some fresh concept name M (again, there are standard API functions: role_fillers and individuals_related?). However, for negated roles, we need to perform an ABox satisfiability (consistency) check, since negated roles are not supported in $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$: $\mathcal{A} \models \Phi(\neg R)_{x \leftarrow i, y \leftarrow j}$ iff $\mathcal{A} \cup \{(i,j) : R\}$ is unsatisfiable. These "reduction tricks" are well known [111]. Particular API functions are called for constraint query atoms as well.

3.5.3 Concrete SUQL Instantiations for the DLMAPS System

We have discussed four representation options in the DLMAPS system. Although the principal ideas have been laid out, we briefly present the resulting spatio-thematic query languages in the SUQL framework for the DLMAPS system. Which spatio-thematic QL is now applicable for the different representation options (1–4) in the DLMAPS system?

Option 1: We can use plain NRQL, as explained.

Option 2: The resulting hybrid QL is called SNRQL. It provides the following additional spatial atoms (note that it does not really add to the message of this text to define these here formally); the extensions of the atoms are computed on the fly by means of inspection methods.

RCC atoms: Atoms such as `(?x ?y (:tppi :ntppi))`; `(:tppi :ntppi)` denotes the disjunctive RCC relation $\{TPPI, NTPPI\}$. A rich set of common sense natural language spatial prepositions such as `:contains, :adjacent, :crosses, :overlaps, :flows-in` is available. The Θ function rewrites these into (the closest possible) RCC relation.

Distance Atoms: `(?x ?y (:inside-distance <min> <max>))`, where `<min>, <max>` specifies an interval $[min; max]$; `NIL` can be used for 0 (or ∞); this applies to the subsequent interval specifications as well. For example, the extension of `(i ?x (:instance-distance nil 100))` consists of all SBox objects which are *not further than 100 meters from* i. Either the shortest distance or the distance between the centroids of these objects is used.

Epsilon Atoms: `(?x ?y (:inside-epsilon <min> <max>))`. With that atom, all objects `?y` are retrieved, such that `?y` is contained within the *buffer zone* of a size specified by the interval $[min; max]$ around `?x`. This buffer zone consists of all points (x, y) whose *shortest distance to the fringe of (the individual bound to)* `?x` is contained within $[min; max]$.

Geometric Attribute Atoms: Atoms regarding geometric attributes, e.g. length and area: The extension of `(?x (:area 100 1000))` consists of all nodes of type polygon in V whose area is in $[100; 1000]$. Also `:length` is understood for linear objects. Moreover, simple type checking atoms such as `(?x :is-polygon)`, `(?x :is-line)` etc. are available (these are needed in order to guard the application of certain spatial operators).

Here is a query which selects an appropriate home for a millionaire:

```
(retrieve (?villa ?living-area  ?golf-club ?church)
  (and (?*living-area (and living-area
                           (or (all classification first-class-area)
                               (string= name "Beverley Hills"))))
       (?living-area ?villa :contains)
       (?*villa (and villa
                     (all status for-sale) (> has-price 10000000)
```

```
                (some has-comfort swimming-pool)))
(?church ?living-area (:inside-epsilon nil 200))
(?living-area ?golf-club :adjacent)
(?*golf-club (and golf-club (all members millionaire)))))
```

Option 3: In principle like SNRQL, but the queries are no longer hybrid. Moreover, the MIDELORA prover currently does not offer concrete domains. Thus, the ABox query language part is reduced to concept and role query atoms.

Option 4: The resulting hybrid QL is called NRQL + *RCC atoms*. This language can only offer RCC atoms in addition to NRQL, since the geometry of the map is not represented. The same syntax is used as for the SNRQL RCC atoms (but the implementation obviously differs, since geometric computations are required in one case, and RCC constraint checking in the other case).

Building OBIS with enabling DL technology is a non-trivial task, especially for IS in non-standard domains. The space of design decisions is very large. Thus we have designed a flexible and generic framework which offers appropriate abstractions that are able to cover *regions* in these design spaces instead of just points.

Since decidability and scalability is not always easy to achieve for OBIS, we believe that it is of even more importance to identify practical solutions which, even though they do not exploit or advance the latest theoretical state-of-the-art techniques in DL research, can nevertheless be considered an advance regarding the current state-of-the-art IS technology and provide guidance and "road maps" for similar designs.

We claim that our framework for building pragmatic combinations of specialized representation layers (including DL ABoxes) for which orthogonal specialized substrate QLs and dedicated provers can be defined, provides a great deal of flexibility for building similar OBIS. Moreover, some of the functionality described here is immediately available for other users (of the RACER system). We claim that the identified software abstractions are valuable, also for non-Lisp developers, as are the identified optimization techniques for ontology query answering engines.

4 Leveraging the Expressivity of Grounded Conjunctive Query Languages

In this section, we present a pragmatic extension of a Semantic Web query language (including so-called grounded conjunctive queries) with a termination safe functional expression language. This addresses problems encountered in daily usage of Semantic Web query languages for which currently no standardized solutions exist, e.g., how to define aggregation operators or expressive procedural predicates, and how to specify so-called combined TBox/ABox queries. We claim that the solution is very flexible, since users can define and execute ad hoc extensions efficiently on the server without having to compile specialized "plugins" in advance. We also address the scalability aspect by showing how efficient aggregation operators can be realized. We claim that the flexibility offered by the approach promises to enhance the applicability of Semantic Web query languages to real world problems.

Nowadays, so-called Description Logics (DLs) provide the basis for SEMANTIC WEB technology, and in particular, for the de facto standards for SEMANTIC WEB ontology languages such as OWL [147]. DL systems can thus be used as SEMANTIC WEB repositories. They offer a set of standard reasoning services, such as consistency checking, automatic computation of the concept (class) hierarchy (the so-called taxonomy), and the basic retrieval services (e.g., instance retrieval) [12]. In the context of the SEMANTIC WEB, especially expressive query languages (which go beyond the basic retrieval services) are of great importance to realize the vision of semantic information retrieval.

In order to enhance the expressivity of SEMANTIC WEB query languages in general and of nRQL (query language of RACER, [155]) in particular, we claim that a kind of server-side programming or procedural extensions are often required. So-called stored procedures are well-known in the relational database realm. However, standard relational database technology is not directly applicable to realize expressive SEMANTIC WEB information retrieval, if expressive background ontologies (e.g., in $\mathcal{SHIQ}$ or OWL) are involved. Moreover, the use of stored procedures can result in unsafe, non-terminating queries. Since decidability is crucial in the SEMANTIC WEB context, we think that any kind of procedural extension facility should be expressive and efficiently computable, but still remain termination safe.

The so-called MiniLisp extension of nRQL is designed to meet these requirements. With MiniLisp a user can write a simple, termination safe, "program" which is executed on the RACER server. Such MiniLisp programs can be used to improve the flexibility of the query language. MiniLisp allows a restricted kind of server-side programming and can thus be used to realize, among others, user-defined query predicates, efficient aggregation operators (e.g., `sum`, `avg`, like in SQL), user-defined output formats for query results, as well as certain kinds of so-called combined ABox/TBox queries, etc. We will present these expressive means in this paper.

4.1 Background: Grounded Conjunctive Queries

In addition to the basic retrieval inference service, expressive query languages are required in practical applications. Well-established is the class of conjunctive queries. A *conjunctive query* consists of a *head* and a *body*. The head lists variables for which the user would like to compute bindings. The body consists of query atoms in which all variables from the head must be mentioned. If the body contains additional variables, they are seen as existentially quantified. A query answer is a set of tuples representing bindings for variables mentioned in the head. A query is a structure of the form $ans(X_1, \ldots, X_n) \leftarrow atom_1, \ldots, atom_m$.

Query atoms can be *concept* query atoms ($C(X)$), *role* query atoms ($R(X, Y)$), *same-as* query atoms ($X = Y$) as well as so-called *concrete domain* query atoms. The latter are introduced to provide support for querying the concrete domain part of a knowledge base. Complex queries are built from query atoms using boolean constructs for conjunction (indicated with comma) or union ($\vee$).

In *standard* conjunctive queries, variables (in the head and in query atoms in the body) are bound to (possibly anonymous) domain objects. In so-called *grounded* conjunctive queries, $C(X)$, $R(X,Y)$ or $X = Y$ are true if, given some bindings α for mapping from variables to *individuals mentioned in the ABox* $\mathcal{A}$, it holds that $(\mathcal{T},\mathcal{A}) \models \alpha(X) : C$, $(\mathcal{T},\mathcal{A}) \models (\alpha(X),\alpha(Y)) : R$, or $(\mathcal{T},\mathcal{A}) \models \alpha(X) = \alpha(Y)$, respectively. In grounded conjunctive queries the standard semantics can be obtained for so-called tree-shaped queries by using corresponding existential restrictions in query atoms [85]. Due to space restrictions, we cannot discuss the details here. In the following, we consider only grounded conjunctive queries, which are implemented by nRQL.

nRQL supports the following atoms for querying the ABox: *concept query atoms*, e.g. `(?x (some has-child human))`; *role query atoms,* e.g. `(?x ?y has-child)`, and (binary) *constraint query atoms*. The following query uses all kinds of nRQL atoms:

```
(retrieve (?x)
   (and (?x (and woman (min age 40))) (?x ?y has-child)
        (?y ?y (constraint (has-father age) (has-mother age)
                           (<= (+ age-2 8) age-1)))))
```

This query returns thus instances of the concept `woman` which are older than 40 and which have children whose fathers are at least 8 years older than their mothers. Note that `(has-father age)` denotes a role chain ended by a so-called concrete domain attribute, a kind of "path expression": starting from the individual bound to `?y` (the child), 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 check whether the predicate `(<= (+ age-2 8) age-1)` holds for them; `age-2` refers to `(has-mother age)`, and `age-1` refers to `(has-father age)`.

In addition to the standard classical negation operator of $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ which can be applied to concepts and also to roles in nRQL, also a so-called negation as failure (NAF) semantics is supported. This means that any of the discussed atoms can appear as NAF-negated in a query body as well.

4.2 Server-Side Programming with Lambda Head Operators

Let us consider an example which demonstrates that server-side programming is sometimes necessary. Consider an ABox representing objects in a geographic information system having width and length, and we want to compute and return the area of these objects with a query. In principle, one can conceive a ternary atom such as `compute-area(?w,?l,?area)`; if `?w` and `?l` are bound to concrete width and length values, then, in order to satisfy the atom, `?area` must be bound to `?w * ?l`. However, the list of such conceivable procedural atoms will be endless and it should thus be possible for users to extend it. Obviously, then either a plug-in mechanism [73] allowing for the specification of user-defined predicates, or a definition language being part of the query language is required. This is the approach we are taking here. The basic idea is simple: In order to keep the semantics of the query body clean and to avoid computational problems (e.g., non-termination of a query caused by a malicious user-defined atom), we are not allowing user-defined procedural atoms in query bodies. Instead, only retrieval conditions can be specified in a body, and in a final mapping step, user-defined projections resp. predicates can be applied to the computed query result tuples.

A so-called (lambda) head projection operator is a function which is applied to the current binding of the variable (the current individual), and the result of this application is included in the answer tuple. The application of a head projection operator can be understood as a function application. So-called

lambda expressions can denote (anonymous) functions. NRQL allows for the specification of lambda expressions in the head of a query; NRQL uses a Lisp dialect which we call MiniLisp. MiniLisp is a termination-safe expression language.

Consider the following ABox in which the individual `box1` is defined, having width of 10 and length of 20:

```
(define-concrete-domain-attribute width :type integer)
(define-concrete-domain-attribute length :type integer)
(instance box1 (and (equal width 10) (equal length 20)))
```

We can then query for the areas of the objects in this ABox as follows:

```
? (retrieve
     (?x
      ((lambda (w l) (* (first w) (first l)))
        (told-value-if-exists (width ?x))
        (told-value-if-exists (length ?x))))
    (?x (and (a width) (a length))))

> (((?x box1) 200))
```

The query body `(?x (and (a width) (a length)))` selects all ABox individuals which have specified fillers of the concrete domain attributes `width` and `length`. However, only in certain cases it is possible to actually retrieve these values. In case such fillers are explicitly known as in our example, the retrieval of concrete values is supported by means of the functional expressions `(told-value-if-exists (width ?x))` (analogously for `height`). We call these expressions projection operators. In general, it is thus also not possible to bind such fillers to variables, e.g., to `?w, ?l`. On the one hand, this is caused by the open world assumption employed in DLs, and is on the other hand a consequence of the concrete domains which allow for the specification of constraint systems that need not have unique solutions. Returning to the example, the function / lambda application is performed by substituting the formal parameters `w, l` with the actual arguments supplied by the two `told-value-if-exists` projection operators. These operators return lists of (told) values; thus, the *first* function is applied before `*` is applied to yield the total area.

MiniLisp is easy to understand and use for readers which have some COMMON LISP experience. In a nutshell, MiniLisp supports numbers, symbols, strings and lists (and thus also trees), offers conditional execution, structure mapping functions (e.g., higher-order functions which apply a function to a list or a tree such as `maptree`), as well as standard functions (e.g., standard arithmetic, list and string processing, comparison and sorting functions) borrowed from the host language COMMON LISP. In order to grant termination, lambdas as required for higher-order functions such as `maptree` etc. are not first order objects. Thus, no unbounded loops can be specified.

Importantly, all RACER API functions can be called from within a lambda body. This also applies to `retrieve` itself, the basic NRQL API query answering function. So, queries can be nested in arbitrary depth, which allows for powerful subquerying. Nested queries allow to achieve a better scalability of expressivity, for example, aggregation operators can be realized in that way (see below). In many cases, inner subqueries shall be constructed based on variable bindings established by outer queries. For this purpose, query templates can be constructed using the backquote-comma template mechanisms from COMMON LISP (similar template construction mechanisms are, for example, known from dynamic web page construction frameworks). We will illustrate this template mechanism in the following sections.

4.3 Predefined Lambda Expressions

Certain commonly used lambda expressions are available as "macros", facilitating idiomatic usage. For example, the RACER API function `direct-types` can be applied to an ABox individual to include the set of direct types the individual is an instance of in the query result. Thus, the lambda expression `((lambda (?x) (direct-types ?x *current-abox*)) ?x))` can be abbreviated as `(direct-types ?)`. Other important operators are `types`, `describe`, and `individual-synonyms`.

For example, on the ABox

```
(instance i c)
```

the subsequent queries return the following results, if `(implies c d)` is in the TBox:

```
? (retrieve (?x (types ?x)) (?x top) :include-lambdas-p t)
> (((?x i)
    (((lambda (?x) (instantiators ?x *current-abox*)) ?x)
     ((c) (d) (*top* top)))))


? (retrieve (?x (all-types-flat ?x)) (?x top)
            :include-lambdas-p t)
> (((?x i)
   (((lambda (?x)
      (sort-symbol-name-lessp
        (flatten (instantiators ?x *current-abox*))))
     ?x)
    ((c d *top* top)))))
```

In the next sections we present several examples which demonstrate the usefulness of MiniLisp for leveraging the expressivity of query languages.

4.4 Aggregation Operators

First we show how queries with *aggregation operators* can be implemented. Consider the following KB which models the compositional structure of a car, see Figure 8. A car has certain parts, and each part has a certain weight:

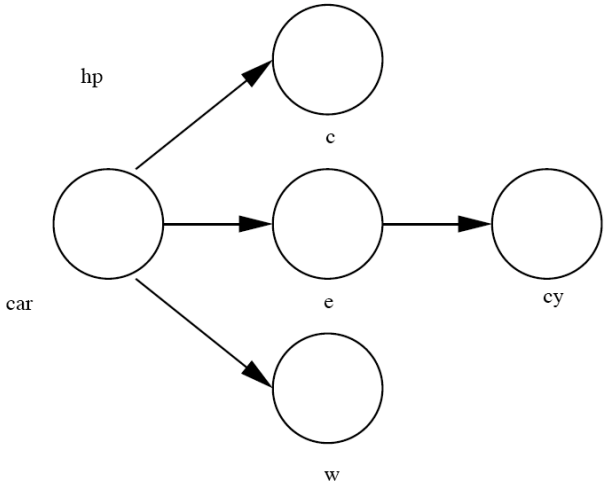

Figure 8: Compositional structure of `mycar`

```
(define-primitive-role has-part :transitive t)
(define-concrete-domain-attribute weight :type real)

(instance mycar car)
(related mycar engine1 has-part)
(related engine1 cylinder-1-4 has-part)
(related mycar wheel-1-4 has-part)
(related mycar chassis1 has-part)
(instance engine1 (= weight 200.0))
(instance chassis1 (= weight 400.0))
(instance wheel-1-4 (= weight 30.0))
```

Using MiniLisp, we can compute the overall weight as well as identify its number of components:

```
? (retrieve1 (?car car)
     (((lambda (car)
        (let ((car-weight
               (reduce '+
                  (flatten
                     (retrieve1 `(and (,car car)
                                      (,car ?part has-part)
                                      (?part (a weight)))
                                `(((lambda (car-weight)
                                     car-weight)
                                   (told-value-if-exists
                                      (weight ?part))))))))
              (car-parts
                (length
                  (retrieve `(?part)
                            `(,car ?part has-part)))))

          `((?car ,car) (?no-of-parts ,car-parts)
            (?total-weight ,car-weight))))

        ?car)))

> ((((?car mycar) (?no-of-parts 4) (?total-weight 630.0))))
```

Please note that `retrieve1` is like `retrieve`, but with head and body argument positions flipped (in order to enhance the "left to right" readability of complex query nestings). The body of the query consists of the concept query atom `(?car car)`. The `lambda` expression is then applied to the current binding of `?car`. So, within the `lambda`, `car` is bound to the binding of `?car`. First, the total weight is computed. For this purpose, a subquery is constructed. If `?car = mycar`, then the query `(retrieve1 `(and (mycar car) (mycar ?part has-part) (?part (a weight))) ...)` is constructed and posed, asking for the parts of `mycar`. The head of the sub query consists of yet another `lambda`, which simply applies the `told-value-if- exists` head projection operator to retrieve the told values of the `weight` attribute of `?part`. The subquery result is returned as a nested list; the list if then flattened, and its items are summed using `(reduce '+ ...)`. The result is bound to the local variable `car-weight`. Similarly, the number of `car-parts` is computed (by posing yet another subquery). Finally, the result of the `lambda` expression is constructed and returned. The constructed and returned value will become the result tuple. So, if `car` is `mycar`, and `no-of-parts` is `4`, and `car-weight` is `630.0`, then the template `` `((?car ,car) (?no-of-parts ,car-parts) (?total-weight ,car-weight)) `` constructs the result tuple `(((?car mycar) (?no-of-parts 4) (?total-weight 630.0)))`.

4.5 Efficient Aggregation Operators using Promises

Although the previous query demonstrated the power and utility of MiniLisp, the aggregation is not computed efficiently. Obviously, this is a problem if the approach is expected to scale. The reason is that for each binding of `?car`, two new subqueries are constructed. Each query has to be parsed, compiled and optimized, and is then maintained as a query object. Since the structures of the subqueries do not change during query execution it would be better to construct these subqueries in advance. This can be achieved using so-called *promises*. A promise declares that certain variables have to be treated as individuals. Such variables are then handled in a very efficient way by the query evaluation engine at runtime. A query containing variables which have been declared by a promise may only be executed if bindings for these variables are established beforehand. A more efficient version of the pervious aggregation query therefore looks as follows:

```
(with-future-bindings (?car)
    (prepare-abox-query (?part)
                  (and (?car car) (?car ?part has-part))
                  :id :parts-of-car-query)

    (prepare-abox-query
                  (((lambda (weight) weight)
                    (told-value-if-exists (weight ?part))))
             (and (?car car)
                  (?car ?part has-part)
                  (?part (a weight)))
             :id :weights-of-parts-of-car-query))
```

This prepares two queries named `:parts-of-car-query` and `:weights-of-parts- of-car-query`. The queries are compiled and optimized, but not executed yet. Since NRQL supports full life cycle management for queries, these queries are from now on available as query objects, ready for execution. Due to the surrounding lexical promise `with-future-bindings`, the query optimizer has treated the `?car` variable as an individual. Thus, we have "promised" NRQL that we will only execute these queries if we supply a binding for `?car` in advance. We can establish such a binding during query execution using `with-nrql-settings` as follows:

```
? (retrieve1 (?car car)
   (((lambda (car)
        (with-nrql-settings (:bindings `((?car ,car)))
           (let ((weight-of-car
                  (reduce '+
                     (flatten
                       (execute-or-reexecute-query
                             :weights-of-parts-of-car-query))))
                 (parts-of-car
                     (length
                          (execute-or-reexecute-query
                             :parts-of-car-query))))

               `((?car ,weight-of-car)
                 (?total-weight ,weight-of-car)))))
        ?car)))

>  ((((?car mycar) (?no-of-parts 4) (?total-weight 630.0))))
```

This results in a very efficient query execution, since (re)execution of a prepared query is immediate (only a function call to the compiled query evaluation function is required).

4.6 Miscellaneous Features

User Defined Query Result Format As already demonstrated, the value returned by a `lambda` body is included in the binding list. In the previous query, the `lambda` body returns a structured list using the template `` `((?car ,car) (?total-weight ,weight-of-car)) ``; this is equivalent to `(list (list '?car car) (list '?total-weight weight-of-car))`. It is also easy to specify natural language output. If we replace `` `((?car ,car) (?total-weight ,weight-of-car)) `` in the previous query with

```
(format nil "Car with name ~A weights ~A kg and has ~A parts."
            car weight-of-car parts-of-car)
```

and execute the query again, then the query result is not returned as a list of variable bindings, but as a list of formatted strings such as

```
"Car with name mycar weights 630.0d0 kg and has 4 parts."
```

Writing Query Results to Files In practical scenarios, file output of query results is important as well. MiniLisp offers the `with-open-output-file` operator which realizes this functionality. Within the dynamic scope of this environment, output to the `*output-stream*` is sent to the specified file. For example, if the previous query format statement is replaced by the

```
(format *output-stream*
        "Car with name ~A weights ~A kg and has ~A parts."
        car weight-of-car parts-of-car)
```

and this is executed in the dynamic scope of an `(with-open-output-file ("minilisp-output.txt") ...)` environment, then the query result will be written to the file `minilisp-output.txt`.

Filtering Result Tuples Sometimes, certain query predicates cannot be specified in the query body due to the "decidability vs. expressivity trade-off" of the $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ concept language. For example, there is no concrete domain predicate searching for substrings, or, returning to our example, it is not possible to exclude all boxes whose area satisfies $width * length \leq 200$. Fortunately, MiniLisp can help here as well, since `lambda` bodies can also work as *filters*. If the special token `:reject` is returned from the `lambda` body, then the result is ignored, i.e., will not appear in the query answer. We can easily identify all parts which have no subparts as follows:

```
? (retrieve1 (?part top)
     (((lambda (part)
        (let ((parts
               (length
                (retrieve `(?part) `(,part ?part has-part)))))
          (if (zerop parts) :reject `(,part has ,parts parts))))
       ?part)))
> (((engine1 has 1 parts)) ((mycar has 4 parts)))
```

Please note that the combination of MiniLisp and `:reject` token gives you the ability to define arbitrary, used-defined *filter predicates* which are executed efficiently since they are directly on the RACER server.

Combined TBox / ABox Queries Finally, let us present an example which demonstrates how to combine TBox queries and ABox queries. Sometimes, one wants to retrieve only the direct instances of a concept / OWL class. An individual is called a direct instance of a concept / OWL class if there is no subconcept / subclass of which the individual is also an instance of.

Let us create two concepts `c` and `d` such that `d` is a subconcept (child concept) of `c`. Thus, the TBox contains

```
(define-concept c (some r top))
(define-concept d (and c e))
```

We can easily verify that `d` has been recognized as a child concept of `c`, using a so-called TBox query:

```
? (tbox-retrieve (?x) (c ?x has-child))
> (((?x d)))
```

Let us create two individuals `i` and `j` in the ABox which are instances of `c`; moreover, `j` is also an instance of `d`:

```
(related i j r)
(related j k r)
(instance j e)
```

We can easily check for the instances of `c` and `d` as follows:

```
? (retrieve (?x) (?x c))
> (((?x j)) ((?x i)))

? (retrieve (?x) (?x d))
> (((?x j)))
```

The previous queries demonstrated that both `i` and `j` are `c` instances. However, only `i` is a direct `c` instance. Unfortunately, retrieval of direct instances is not a standard DL inference problem; thus, there is no RACER API function. But using a combined ABox/TBox query it is indeed possible to retrieve the direct instances of `c` as follows:

```
? (retrieve1 (?x c)
     (((lambda (x)
        (if (some (lambda (subclass)
                    (retrieve () `(,x ,subclass)))
                  (flatten
                    (tbox-retrieve1 `(c ?subclass has-child)
                      `(((lambda (subclass) subclass) ?subclass)))))
                  :reject
                `(?x ,x)))
        ?x)))
> (((?x i)))
```

In this query, `?x` is bound to a `c` instance. Using this binding, it is checked by means of a TBox subquery (`tbox-retrieve1`) whether the individual bound to `?x` is also an instance of any subclass of `c`. If this is the case, the result tuple (resp. the current binding of `?x`) is *rejected* (see the special `:reject` token); otherwise, the result tuple is *constructed* and returned. The returned result tuples make up the final result set.

4.7 Conclusion

We have presented a pragmatic extension of a SEMANTIC WEB query language (including so-called grounded conjunctive queries) by lambda expressions. The termination safe functional expression language MiniLisp offers solutions to problems encountered in daily usage of SEMANTIC WEB query languages for which currently no standardized solutions exist. nRQL is the first and only SEMANTIC WEB query language that offers used-defined ad hoc predicates, aggregation operators, as well as combined ABox+TBox queries. Moreover, the proposed solution is technically sound, since the query body is kept clean from user defined predicates which might result in unsafe queries. The solution is very flexible, since users can define define and execute ad hoc extensions efficiently on the server without having to compile specialized "plugins" in advance. We have also addressed the scalability aspects by showing how efficient aggregation operators can be realized in this framework by exploiting the notion of promises. Standard aggregation operators could also be made accessible as macros (idioms). We believe that the flexibility offered by MiniLisp greatly enhances the applicability of the query language to real world problems.

5 Unrestricted Conjunctive Queries for Expressive Description Logics

5.1 Introduction

Description Logics (DLs) [8] are a well-established family of logic-based knowledge representation formalisms that have recently gained increased attention due to their usage as the logical underpinning of ontology languages such as DAML+OIL and OWL [83]. A DL knowledge base (KB) consists of a TBox, which contains intensional knowledge such as concept definitions and general background knowledge, and an ABox, which contains extensional knowledge and is used to describe individuals. Using a database metaphor, the TBox corresponds to the schema, and the ABox corresponds to the data.

In data-intensive applications, querying KBs plays a central role. Essentially, there are two forms of querying. The first one is *instance retrieval*, which allows the retrieval of all certain instances of a given (possibly complex) concept C, i.e., it returns all individuals from the ABox that are an instance of C in every model of the KB. Technically, instance retrieval is well-understood. For the prominent DL $\mathcal{SHIQ}$, which underlies DAML+OIL and OWL Lite, it is EXPTIME-complete [143], and, despite this high worst-case complexity, efficient implementations are available. On the other hand, instance retrieval is a rather poor form of querying: concepts are used as queries, and thus we can only query for structures that are invariant under (guarded) bisimulations. For this reason, many applications require *conjunctive query answering* as a stronger form of querying, i.e., computing the certain answers to a conjunctive query over a DL knowledge base.

Until now it was an open problem whether conjunctive query answering is decidable in $\mathcal{SHIQ}$. In particular, the presence of transitive and inverse roles makes the problem rather tricky [55], and results were only available for two restricted cases. The first case is obtained by stipulating that the variables in queries can only be bound to individuals that are explicitly mentioned in the ABox. The result is a form of closed-domain semantics, which is different from the usual open-domain semantics in DLs. It is easily seen that conjunctive query answering in this setting can be reduced to instance retrieval. In the second case, the binary atoms in conjunctive queries are restricted to roles that are neither transitive nor have transitive sub-roles, and it is known that conjunctive query answering in this setting is decidable [37, 44] and co-NP-complete regarding data complexity [119].

In this section, we show that unrestricted conjunctive query answering in $\mathcal{SHIQ}$ is decidable. More precisely, we devise a decision procedure for the *entailment* of a conjunctive query by a $\mathcal{SHIQ}$ knowledge base, which is the decision problem corresponding to conjunctive query answering. It is well-known that decidability and complexity results carry over from entailment to answering. Our decision procedure for query entailment consists of a rather intricate reduction to KB consistency in $\mathcal{SHIQ}^{\sqcap}$, i.e., $\mathcal{SHIQ}$ extended with role conjunction. The latter is easily seen to be decidable. The resulting (deterministic) algorithm for conjunctive query entailment in $\mathcal{SHIQ}$ needs time double exponential in the size of the query and single exponential in the size of the KB. This result concerns the combined complexity, i.e., it is measured in the size of all inputs: the query, the ABox, and the TBox. Since query and TBox are usually small compared to the ABox, the data complexity (measured in the size of the ABox, only) is also a relevant issue. We show that (the decision problem corresponding to) conjunctive query answering in $\mathcal{SHIQ}$ is co-NP-complete regarding data complexity, and thus not harder than instance retrieval [86].

5.2 Conjunctive Queries

Let $\mathsf{N_V}$ be a countably infinite set of *variables* disjoint from $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$. An *atom* is an expression $A(v)$ (*concept atom*) or $r(v, v')$ (*role atom*), where A is a concept name, 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 its elements. We use $\mathsf{Var}(q)$ ($\mathsf{Var}(at)$) to denote the set of variables occurring in the query q (atom at). Let $\mathcal{I}$ be an interpretation, q a conjunctive query, and $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ a total function. We write

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

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

The *query entailment problem* is defined as follows: given a knowledge base $\mathcal{K}$ and a query q, decide whether $\mathcal{K} \models q$. It is well-known that query entailment and query answering can be mutually reduced and that decidability and complexity results carry over [31, 76]. In what follows, we concentrate on query entailment.

For convenience, we assume that conjunctive queries are closed under inverses, i.e., if $r(v, v') \in q$, then $\mathsf{Inv}(r)(v', v) \in q$. If we add or remove atoms from a query, we silently assume that we do this such that the resulting query is again closed under inverses. We will also assume that queries are connected. Formally, a query q (closed under inverses) is *connected* if, for all $v, v' \in \mathsf{Var}(q)$, there exists a sequence $v_0, \ldots, v_n$ such that $v_0 = v$, $v_n = v'$, and for all $i < n$, there exists a role r such that $r(v_i, v_{i+1}) \in q$. A collection $q_0, \ldots, q_k$ of queries is a *partitioning* of q if $q = q_0 \cup \cdots \cup q_k$, $\mathsf{Var}(q_i) \cap \mathsf{Var}(q_j) = \emptyset$ for $i < j \leq k$, and each q_i is connected. The following lemma shows that connectedness can be assumed w.l.o.g.

Lemma 8. *Let $\mathcal{K}$ be a knowledge base, q a conjunctive query, and $q_0, \ldots, q_n$ a partitioning of q. Then $\mathcal{K} \models q$ iff $\mathcal{K} \models q_i$ for all $i \leq n$.*

5.3 Forests and Trees

In this section, we carefully analyze the entailment of queries by knowledge bases and establish a set of general properties that will play a central role in our decision procedure. We start by showing that, in order to decide whether $\mathcal{K} \models q$, it suffices to check whether $\mathcal{I} \models q$ for all models $\mathcal{I}$ of $\mathcal{K}$ that are of a particular shape. Intuitively, these models are shaped like a forest (in the graph-theoretic sense), modulo the fact that transitive roles have to be interpreted in transitive relations.

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 w is a successor of w'.

Definition 9. 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 $a \in \mathsf{Ind}(\mathcal{A})$, $a^{\mathcal{I}} = (b, \varepsilon)$ for some $b \in \mathsf{Ind}(\mathcal{A})$.

A model $\mathcal{I}$ of $\mathcal{K}$ is *canonical* 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, in canonical models $\mathcal{I}$, each individual a is mapped to a pair (b, ε), where $a = b$ does not necessarily hold. We need this since we do not adopt the *uniqe name assumption (UNA)*: if $a, b \in \mathsf{N_I}$ with $a \neq b$, then we allow that $a^{\mathcal{I}} = b^{\mathcal{I}}$. If desired, the UNA can easily be adopted by adding an assertion $a \neq b$ for each pair of individual names in $\mathsf{Ind}(\mathcal{A})$ to the ABox.

Lemma 10. *$\mathcal{K} \not\models q$ iff there exists a canonical model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models q$.*

Lemma 10 shows that, when deciding whether $\mathcal{K} \models q$, it suffices to check whether $\mathcal{I} \models q$ for all canonical models $\mathcal{I}$ of $\mathcal{K}$. As a next step, we would like to show that, for canonical models $\mathcal{I}$, to check whether $\mathcal{I} \models q$, we can restrict our attention to a particular kind of match π for $\mathcal{I}$ and q. A match π for $\mathcal{I}$ and q is a *forest match* if, for all $r(v, v') \in q$, we have one of the following:

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

Alas, it is not sufficient to only consider forest matches for $\mathcal{I}$ and q. Instead, we show the following: we can rewrite q into a set of queries Q such that, for all canonical models $\mathcal{I}$, we have that $\mathcal{I} \models q$ iff $\mathcal{I} \models^{\pi} q'$ for some $q' \in Q$ and forest match π. Intuitively, this complication is due to the presence of transitive roles.

Definition 11. A query q' is called a *transitivity rewriting* of q w.r.t. $\mathcal{K}$ 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 $s_0, \ldots, s_n \in \mathsf{Trans}_{\mathcal{H}}$ such that $s_i \sqsubseteq^*_{\mathcal{H}} r_i$ for all $i \leq n$, and then replacing $r_i(v_i, v_i')$ with

$$s_i(v_i, u_i), s_i(u_i, v_i')$$
or
$$s_i(v_i, u_i), s_i(u_i, u_i'), s_i(u_i', v_i')$$

for all $i \leq n$, where $u_i, u_i' \notin \mathsf{Var}(q)$. We use $\mathsf{tr}_{\mathcal{K}}(q)$ to denote the set of all transitivity 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 are neglected.

Together with Lemma 10, the following lemma shows that, on canonical models, we have $\mathcal{I} \models q$ iff there is a forest match π and a $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ such that $\mathcal{I} \models^{\pi} q'$.

Lemma 12. *Let $\mathcal{I}$ be a model of $\mathcal{K}$.*

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 $\mathcal{I} \models q$.*

The most important property of forest matches is the following: if $\mathcal{I} \models^{\pi} q$ with π a forest match, then π splits the query q into several subqueries: the *base subquery* q_0 contains all role atoms that are matched to root nodes:

$$q_0 := \{r(v, v') \in q \mid \pi(v), \pi(v') \in \mathsf{N_I} \times \{\varepsilon\}\};$$

Moreover, for each $(a, \varepsilon) \in \mathsf{N_I} \times \{\varepsilon\}$ which occurs in the range of π, there is an *object subquery* q_a:

$$q_a := \{at \mid \forall v \in \mathsf{Var}(at) : \pi(v) \in \{a\} \times \mathbb{N}^*\} \setminus q_0.$$

Clearly, $q = q_0 \cup \bigcup_a q_a$. Although the resulting subqueries are not a partitioning of q in the sense of Section 2, one of the fundamental ideas behind our decision procedure is to treat the different subqueries more or less separately. The main benefit is that the object subqueries can be rewritten into tree-shaped queries which can then be translated into concepts. This technique is also known as "rolling up" of tree conjunctive queries into concepts and was proposed in [31, 76]. Formally, a query q is *tree-shaped* if there exists a bijection σ from $\mathsf{Var}(q)$ into a tree T such that $r(v, v') \in q$ implies that $\sigma(v)$ is a neighbor of $\sigma(v')$ in T. Before we show how to rewrite the object subqueries into tree-shaped queries, let us substantiate our claim that tree-shaped queries can be rolled up into concepts. The result of rolling up is not a $\mathcal{SHIQ}$-concept, but a concept formulated in $\mathcal{SHIQ}^{\sqcap}$, the extension of $\mathcal{SHIQ}$ with role intersection. More precisely, $\mathcal{SHIQ}^{\sqcap}$ is obtained from $\mathcal{SHIQ}$ by admitting the concept constructors $\exists\alpha.C$, $\forall\alpha.C$, $\leqslant \alpha.C$, and $\geqslant \alpha.C$, where α is a role conjunction $r_1 \sqcap \cdots \sqcap r_k$ with the r_i (possibly inverse) roles.

Let q be a tree-shaped query and σ a bijection from $\mathsf{Var}(q)$ into a tree T. Inductively assign to each $v \in \mathsf{Var}(q)$ a $\mathcal{SHIQ}^{\sqcap}$-concept $C_q(v)$:

- if $\sigma(v)$ is a leaf of T, then $C_q(v) := \bigsqcap_{A(v)\in q} A$
- if $\sigma(v)$ has successors $\sigma(v_1), \ldots, \sigma(v_n)$ in T, then

$$C_q(v) := \bigsqcap_{A(v)\in q} A \sqcap \bigsqcap_{1\leq i\leq n} \exists\big(\bigsqcap_{r(v,v_i)\in q} r\big).C_q(v_i).$$

Then the rolling up C_q of q is defined as $C_q(v_r)$, where v_r is such that $\sigma(v_r) = \varepsilon$. (Recall that σ is a bijection, hence, such a v_r exists.) The following lemma shows the connection between the query and the rolled up concept.

Lemma 13. *Let q be a tree-shaped query and $\mathcal{I}$ an interpretation. Then $\mathcal{I} \models q$ iff $C_q^{\mathcal{I}} \neq \emptyset$.*

We now show how to transform a query q into a set Q of tree-shaped ones. To describe the exact goal of this translation, we need to introduce tree matches as a special case of forest matches: a match π for a canonical model $\mathcal{I}$ and q is a *tree match* if the range of π is a subset of $\{a\} \times \mathbb{N}^*$, for some $a \in \mathsf{N_I}$. Now, our tree transformation should be such that

> $(*)$ whenever $\mathcal{I} \models^{\pi} q$ with $\mathcal{I}$ a canonical model and π a tree match, then $\mathcal{I} \models^{\pi'} q'$ for some (tree-shaped) query $q' \in Q$ and tree match π'.

Recall the splitting of a query into a base subquery and a set of object subqueries q_a, induced by a forest match π. It is not hard to see that for each q_a, the restriction of π to $\mathsf{Var}(q_a)$ is a tree match for $\mathcal{I}$ and q_a. Thus, object subqueries together with their inducing matches π satisfy the precondition of $(*)$.

The rewriting of a query into a tree-shaped one is a three stage process. In the first stage, we derive a *collapsing* q_0 of the original query 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 certain existing role atoms $r(v, v')$ redundant, 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 atoms $s(v_0, v_1), \ldots, s(v_{n-1}, v_n) \in q$ with $v_0 = v$, $v_n = v'$, $s \sqsubseteq^* r$, and $s \in \mathsf{Trans}$. Combining the extension and reduct steps allows us, e.g., to transform a non-tree-shaped "loop" $r(v, v)$ into a tree shape by adding 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 $r(v, v)$.

In what follows, the *size* $|q|$ of a query q is defined as the number of atoms in q.

Definition 14. A *collapsing* of q is obtained by identifying variables in q. A query q' is an *extension* of q w.r.t. $\mathcal{K}$ if:

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 $\mathcal{K}$;
4. $|\mathsf{Var}(q')| \leq 4|q|$;
5. $|\{r(v, v') \in q' \mid r(v, v') \notin q\}| \leq 171|q|^2$.

A query q' is a *reduct* of q w.r.t. $\mathcal{K}$ if:

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 query q' is a *tree transformation* of q w.r.t. $\mathcal{K}$ if there exist 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. $\mathcal{K}$, and
- q' is a tree-shaped reduct of q_1.

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

We remark that Condition 5 of query extensions is not strictly needed for correctness, but it ensures that our algorithm is only single exponential in the size of $\mathcal{K}$. As in the case of $\mathtt{tr}_{\mathcal{K}}(q)$, we assume that $\mathtt{tt}_{\mathcal{K}}(q)$ does not contain any isomorphic queries.

The next lemma states the central properties of tree transformations. Before we can formulate it, we introduce two technical notions. Let $q' \in \mathtt{tt}_{\mathcal{K}}(q)$, $\mathcal{I} \models^{\pi} q$, and $\mathcal{I} \models^{\pi'} q'$. Then π and π' are called *ε-compatible* if the following holds: if $v \in \mathsf{Var}(q)$, v was identified with variable $v' \in \mathsf{Var}(q')$ during the collapsing step, and at least one of $\pi(v), \pi'(v')$ is in $\mathsf{N_I} \times \{\varepsilon\}$, then $\pi(v) = \pi'(v')$. Furthermore, we call π an *a-tree match* if $\pi(v) \in \{a\} \times \mathbb{N}^*$ for all $v \in \mathsf{Var}(q)$.

Lemma 15. *Let $\mathcal{I}$ be a model of $\mathcal{K}$.*

1. *If $\mathcal{I}$ is canonical and π an a-tree match with $\mathcal{I} \models^{\pi} q$, then there is a $q' \in \mathtt{tt}_{\mathcal{K}}(q)$ and an a-tree match π' such that $\mathcal{I} \models^{\pi'} q'$ and π, π' are ε-compatible.*
2. *If $q' \in \mathtt{tt}_{\mathcal{K}}(q)$ and $\mathcal{I} \models^{\pi'} q'$, then there is a match π with $\mathcal{I} \models^{\pi} q$ and π, π' are ε-compatible.*

Intuitively, using a-tree matches and ε-compatibility in Lemma 15 ensures that, if we are given a match for the base subquery and a tree match for a tree transformation of each object subquery, then we can construct a forest match of the original query.

5.4 The Decision Procedure

Let $\mathcal{K}$ be a knowledge base and q a conjunctive query such that we want to decide whether $\mathcal{K} \models q$. A *counter model* for $\mathcal{K}$ and q is a model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models q$. In the following, we show how to convert $\mathcal{K}$ and q into a sequence of knowledge bases $\mathcal{K}_1, \ldots, \mathcal{K}_\ell$ such that (i) every counter model for $\mathcal{K}$ and q is a model of some $\mathcal{K}_i$, (ii) every canonical model of a $\mathcal{K}_i$ is a countermodel for $\mathcal{K}$ and q, and (iii) each consistent $\mathcal{K}_i$ has a canonical model. Thus, $\mathcal{K} \models q$ iff all the $\mathcal{K}_i$ are inconsistent. This gives rise to two decision procedures: a deterministic one in which we enumerate all $\mathcal{K}_i$ and which we use to derive an upper bound for combined complexity; and a non-determinstic one in which we guess a $\mathcal{K}_i$ and which yields a tight upper bound for data complexity.

Since the knowledge bases $\mathcal{K}_i$ involve concepts obtained by rolling up tree-shaped queries, they are fomulated in $\mathcal{SHIQ}^\sqcap$ rather than in $\mathcal{SHIQ}$. More precisely, each KB $\mathcal{K}_i$ is of the form $(\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}_i)$, where

- $(\mathcal{T}, \mathcal{H}, \mathcal{A})$ is a $\mathcal{SHIQ}$ knowledge base;
- $\mathcal{T}_q$ is a set of GCIs $\top \sqsubseteq C$ with C a $\mathcal{SHIQ}^\sqcap$ concept;
- $\mathcal{A}_i$ is a generalized $\mathcal{SHIQ}^\sqcap$-ABox[12] such that $\mathsf{Ind}(\mathcal{A}_i) \subseteq \mathsf{Ind}(\mathcal{A})$.

In what follows, we call knowledge bases of this form *extended* knowledge bases. Using a standard unravelling argument, it is easy to establish Property (iii) from above, i.e., every consistent extended knowledge base $\mathcal{K}$ has a canonical model.

The actual construction of the extended knowledge bases is based on the analysis in Section 5.3. To start with, Lemma 10 and 12 imply the following: to ensure that a canonical model of an extended KB is a counter model for $\mathcal{K}$ and q, it suffices to prevent forest matches of all queries $q' \in \mathsf{tr}_\mathcal{K}(q)$. In order to prevent such matches, we use the parts $\mathcal{T}_q$ and $\mathcal{A}_i$ of extended knowledge bases.

More precisely, we distinguish between two kinds of forest matches: tree matches and *true* forest matches, i.e., forest matches that are not tree matches. Preventing tree matches of a $q' \in \mathsf{tr}_\mathcal{K}(q)$ in a canonical model is relatively simple: by Lemmas 15 and 13, it suffices to ensure that, for all $q'' \in \mathsf{tt}_\mathcal{K}(q')$, the corresponding concept $C_{q''}$ does not have any instances. Therefore, $\mathcal{T}_q$ is defined as follows:

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

It is easily seen that true forest matches π always involve at least one ABox individual (i.e., $\pi(v) \in \mathsf{N_I} \times \{\varepsilon\}$ for at least one variable v). In order to prevent true forest matches, we thus use an ABox $\mathcal{A}_i$. Intuitively, we obtain the ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$ by considering all possible ways of adding assertions to $\mathcal{K}$ such that, for all queries $q' \in \mathsf{tr}_\mathcal{K}(q)$, all true forest matches are prevented. This gives rise to the knowledge bases $\mathcal{K}_1, \ldots, \mathcal{K}_\ell$.

As suggested in Section 5.3, we can prevent a true forest match π of $q' \in \mathsf{tr}_\mathcal{K}(q)$ by splitting π into a base subquery and a number of object subqueries and then making sure that either the base query fails to match (this involves only individual names from the ABox) or at least one of the object subqueries fails to have a tree match. In Section 5.3, however, we used a concrete forest match π to split a query into subqueries. Here, we do not have a concrete π available and must consider *all possible ways* in which a forest match can split a query.

[12] Recall that an ABox is *generalized* if it admits assertions $C(a)$ with C an arbitrary concept.

Let $q' \in \mathsf{tr}_{\mathcal{K}}(q)$. A *splitting candidate* for q' is a partial function $\tau : \mathsf{Var}(q') \to \mathsf{Ind}(\mathcal{A})$ with non-empty domain. For $a \in \mathsf{Ind}(\mathcal{A})$, 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 $v_0, \ldots, v_n$, $n \geq 0$, such that

- $\tau(v_0) = a$ and $v_n = v$;
- for all $i \leq n$, $\tau(v_i) = a$ or $\tau(v_i)$ is undefined;
- for all $i < n$, there is a role r s.t. $r(v_i, v_{i+1}) \in q'$.

We call τ a *split mapping* for q' if, for all $a, b \in \mathsf{Ind}(\mathcal{A})$, $a \neq b$ implies $\mathsf{Reach}(a, \tau) \cap \mathsf{Reach}(b, \tau) = \emptyset$. Intuitively, each split mapping τ for q' represents the set of forest matches π of q' such that $\pi(v) = (\tau(v), \varepsilon)$ for all v in the domain of τ. Each injective split mapping for q' induces a splitting of q' into a base query and object queries. Split mappings τ need not be injective, however, and thus the general picture is that they induce a splitting of the collapsing q'' of q' obtained by identifying all variables v, v' with $\tau(v) = \tau(v')$. This splitting is as follows:

$$\begin{array}{rcl} q_0^\tau & := & \{r(v, v') \in q'' \mid \tau(v), \tau(v') \text{ is defined}\} \\ q_a^\tau & := & \{at \in q'' \mid \mathsf{Var}(at) \subseteq \mathsf{Reach}(a, \tau)\} \setminus q_0^\tau \end{array}$$

for all $a \in \mathsf{N_I}$ that are in the range of τ. Observe that the condition which distinguishes splitting candidates and split mappings ensures that $a \neq b$ implies $\mathsf{Var}(q_a^\tau) \cap \mathsf{Var}(q_b^\tau) = \emptyset$. This condition is satisfied by the splittings described in Section 3, and it is needed to independently roll up the subqueries q_a^τ into concepts.

In the following, we use $\mathsf{sub}(q)$ to denote the set of subqueries of q, i.e., the set of non-empty subsets of q. Let

$$Q := \{q_3 \,|\, \exists q_1, q_2 \colon q_1 \in \mathsf{tr}_{\mathcal{K}}(q), q_2 \in \mathsf{sub}(q_1), q_3 \in \mathsf{tt}_{\mathcal{K}}(q_2)\}$$

and let $\mathsf{cl}(q)$ be the set of rolled up concepts $C_{q'}$, for all $q' \in Q$. A $\mathcal{SHIQ}^\sqcap$ 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 Q and $a, b \in \mathsf{Ind}(\mathcal{A})$.

Let τ be a split mapping for $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ and $\mathcal{A}'$ a q-completion. We say that $\mathcal{A}'$ *spoils* τ if one of the following holds:

(a) there is an $r(v, v') \in q_0^\tau$ such that $\neg r(\tau(v), \tau(v')) \in \mathcal{A}'$;

(b) there is an a in the range of τ such that $\neg C(a) \in \mathcal{A}'$, where

$$C := \bigsqcup_{q'' \in \mathsf{tt}_{\mathcal{K}}(q_a^\tau)} C_{q''}.$$

Intuitively, (b) prevents tree matches of the object subqueries, c.f. Lemmas 15 and 13.

Finally, a q-completion $\mathcal{A}'$ is called a *counter canidate* for $\mathcal{K}$ and q if for all $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ and all split mappings τ for q', $\mathcal{A}'$ spoils τ. Let $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$ be all counter candidates for $\mathcal{K}$ and q and $\mathcal{K}_1, \ldots, \mathcal{K}_\ell$ the associated extended KBs.

Lemma 16. *$\mathcal{K} \models q$ iff $\mathcal{K}_1, \ldots, \mathcal{K}_\ell$ are inconsistent.*

We now define the two decision procedures for query entailment in $\mathcal{SHIQ}$: in the deterministic version, we generate all q-completions $\mathcal{A}'$ of $\mathcal{A}$ and return "$\mathcal{K} \models q$" if all generated $\mathcal{A}'$ that are a counter candidate give rise to an inconsistent extended KB. Otherwise, we return "$\mathcal{K} \not\models q$". In the non-deterministic version, we guess a q-completion $\mathcal{A}'$ of $\mathcal{A}$, return "$\mathcal{K} \not\models q$" if $\mathcal{A}'$ is a counter candidate and the associated extended KB is consistent, and "$\mathcal{K} \models q$" otherwise. To show that these algorithms are decision procedures, it remains to show that the consistency of extended knowledge bases is decidable. The following theorem can be proved by a reduction to the DL $\mathcal{ALCQIb}$ and using results from [143]. In the following, the *size* $|\Gamma|$ of an ABox, TBox, role hierarchy or (extended) knowledge base Γ is simply the number of symbols needed to write Γ (with numbers written in binary).

Theorem 17. *There is a polynomial p such that, given an extended knowledge base $\mathcal{K}' = (\mathcal{T} \cup \mathcal{T}_q, \mathcal{H}, \mathcal{A} \cup \mathcal{A}')$ with $|\mathcal{K}'| = k$, $|\mathcal{A} \cup \mathcal{A}'| = a$, $|\mathcal{T} \cup \mathcal{T}_q \cup \mathcal{H}| = t$, and where the maximum length of concepts in $\mathcal{T}_q$ and $\mathcal{A}'$ is ℓ, we can decide consistency of $\mathcal{K}'$ in*

- *deterministic time in $2^{p(k)2^{p(\ell)}}$;*
- *non-deterministic time in $p(a) \cdot 2^{2^{p(t)}}$.*

We now discuss the complexity of our algorithms. We start by establishing some bounds on the number and size of transitivity rewritings, tree transformations, etc.

Lemma 18. *Let $|q| = n$ and $|\mathcal{K}| = m$. Then there is a polynomial p such that*
(a) $|\mathsf{tr}_{\mathcal{K}}(q)| \leq 2^{p(n)\cdot \log p(m)}$;
(b) *for all $q' \in \mathsf{tr}_{\mathcal{K}}(q)$, $|q'| \leq 3n$;*
(c) *for all $q' \in \mathsf{tr}_{\mathcal{K}}(q)$, $|\mathsf{tt}_{\mathcal{K}}(q')| \leq 2^{p(n)\cdot \log p(m)}$;*
(d) *for all $q' \in \mathsf{tr}_{\mathcal{K}}(q)$ and $q'' \in \mathsf{tt}_{\mathcal{K}}(q')$, $|q''| \leq p(n)$;*
(e) $|\mathsf{cl}(q)| \leq 2^{p(n)\cdot \log p(m)}$; *and*
(f) *for all $C \in \mathsf{cl}(q)$, $|C| \leq p(n)$.*

Let $k = |\mathsf{cl}(q)|$. We first show that the deterministic version of our algorithm runs in time exponential in m and double exponential in n. This follows from Theorem 17 together with the following observations:

(i) The number of q-completions is bounded by $2^{k\cdot m + k\cdot m^2}$, which, by Lemma 18(e), is exponential in m and double exponential in n;

(ii) Checking whether a q-completion is a counter candidate can be done in time exponential in n and polynomial in m;

(iii) By Lemma 18, the cardinality of $\mathcal{T}_q$ and of each q-completion is exponential in n and polynomial in m, and the maximum length of concepts in $\mathcal{T}_q$ and $\mathcal{A}'$ is polynomial in n (and independent of m).

Now for the non-deterministic version. Since we aim at an upper bound for data complexity, we only need to verify that the algorithm runs in time polynomial in the size of $|\mathcal{A}|$, and can neglect the contribution of $\mathcal{T}$, $\mathcal{H}$, and q to time complexity. The desired result follows from Theorem 17 and Points (ii) and (iii) above. This bound is tight since conjunctive query entailment is already co-NP-hard regarding data complexity in the $\mathcal{AL}$ fragment of $\mathcal{SHIQ}$. Summing up, we thus have the following.

Theorem 19. *Conjunctive query entailment in $\mathcal{SHIQ}$ is data complete for co-*NP*, and can be decided in time exponential in the size of the knowledge base and double exponential in the size of the query.*

6 Answering Regular Path Queries in Expressive Description Logics: An Automata-Theoretic Approach

6.1 Introduction

Description Logics (DLs) [13] are a well-established branch of logics for knowledge representation and reasoning, and the premier logic-based formalism for modeling concepts (i.e., classes of objects) and roles (i.e., binary relationships between classes). They have gained increasing attention in different areas including the Semantic Web, data and information integration, peer-to-peer data management, and ontology-based data access. In particular, some of the standard Web ontologies from the OWL family are based on DLs [70].

In DLs, traditionally reasoning tasks had been studied that deal with taxonomic issues like classification and instance checking. Recently, however, the widening range of applications has led to extensive studies of answering queries over DL knowledge bases (KBs) that require, beyond simple instance retrieval, to join pieces of information in finding the answer. Specifically, *conjunctive queries* have been studied in several papers, cf. [37, 44, 35, 87, 88, 120, 57].

As shown therein, answering (classes of) conjunctive queries is decidable for several DLs, including expressive ones. [57] proved this for arbitrary conjunctive queries over $\mathcal{SHIQ}$ KBs, while [87, 88] showed this for conjunctive queries without transitive roles and [120] for unions of such queries.[13]

At present, (unions of) conjunctive queries over $\mathcal{SHIQ}$ KBs is among the most expressive decidable settings. In this paper, we push the frontier and establish decidability of query answering for the yet more expressive class of *positive (existential) two-way regular path queries* (in short, P2RPQs) over the expressive DL $\mathcal{ALCQIb}_{reg}$, which is close to $\mathcal{SHIQ}$. P2RPQs are queries inductively built, using conjunction and disjunction, from atoms that are regular expressions over direct and inverse roles (and allow for testing of concepts). They not only subsume conjunctive queries and unions of conjunctive queries, but also unions of conjunctive regular path queries [43].

More specifically, we make the following contributions.

• Different from previous works, which rely on resolution-based transformations to disjunctive datalog or on tableaux-based algorithms, we use automata techniques for query answering in expressive DLs. While the application of automata techniques in DLs is not novel, cf. [40, 143], previous work was concerned with deciding satisfiability of a KB consisting of a TBox only. Here we address the much more involved task of query answering over a KB, which has data in an ABox; incorporating the query is non-obvious.

• The technique we apply is more accessible than the existing ones based on tableaux and resolution. Indeed, it is computational in nature, and directly works on the models of the KB. In this way, we are also able to obtain more general results, which seems more difficult using the other approaches.

• As a first result, we present an automata-based algorithm for checking the satisfiability of a KB (consisting of TBox and ABox) in EXPTIME. This is worst-case optimal.

• Our main result then shows that answering positive existential queries over $\mathcal{ALCQIb}_{reg}$ KBs is feasible in 2EXPTIME. By a reduction of $\mathcal{SHIQ}$ to $\mathcal{ALCQIb}_{reg}$, a similar result follows for $\mathcal{SHIQ}$. This compares well to the N3EXPTIME bound for union of conjunctive queries in [120], and the 2EXPTIME bounds for (classes of) conjunctive queries that emerge from [57, 87]. On the other hand, we establish an EXPSPACE lower bound for positive existential queries.

Our results indicate that automata-techniques have high potential for advancing the decidability frontier of query answering over expressive DLs, and are a useful tool for analyzing its complexity.

[13] Note that the technique in [37] for unions of conjunctive regular path queries is actually incomplete.

6.2 Preliminaries

6.2.1 $\mathcal{ALCQIb}_{reg}$

Concepts and roles in $\mathcal{ALCQIb}_{reg}$ obey the following syntax:

$$\begin{array}{rcl} C, C' & \longrightarrow & A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \forall R.C \mid \\ & & \exists R.C \mid \geqslant n\, Q.C \mid \leqslant n\, Q.C \\ Q, Q' & \longrightarrow & P \mid P^- \mid Q \cap Q' \mid Q \cup Q' \mid Q \setminus Q' \\ R, R' & \longrightarrow & Q \mid R \cup R' \mid R \circ R' \mid R^* \mid id(C) \end{array}$$

where A denotes an *atomic concept*, P an *atomic role*, C an arbitrary *concept*, and R an arbitrary *role*. We use Q to denote *basic roles*, which are those roles which may occur in number restrictions. W.l.o.g., we assume that "\" is applied only to atomic roles and their inverses.

An $\mathcal{ALCQIb}_{reg}$ *knowledge base* (KB) is a pair $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ where $\mathcal{A}$ (the *ABox*) is a set of *assertions* of the form $A(a)$, $P(a, b)$, and $a \neq b$, with A an atomic concept, P an atomic role, and a, b *individuals*; and $\mathcal{T}$ (the *TBox*) is a set of *concept inclusion axioms* $C \sqsubseteq C'$ for arbitrary concepts C and C'. W.l.o.g. all concepts occurring in $\mathcal{A}$ occur in $\mathcal{T}$. We denote by $\mathcal{C}_\mathcal{K}$ the set of atomic concepts occurring in $\mathcal{K}$, by $\mathcal{R}_\mathcal{K}$ the set of atomic roles occurring in $\mathcal{K}$ and their inverses, and by $\mathcal{J}_\mathcal{K}$ the individuals in $\mathcal{K}$.

The semantics is the standard one [13]. We note that we do *not* adopt the *unique name assumption*.

KB satisfiability consists in determining whether some interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ satisfies all assertions in $\mathcal{A}$ and all concept inclusion axioms in $\mathcal{T}$. By internalization [134], this is reducible to finding an interpretation $\mathcal{I}$ satisfying $\mathcal{A}$ and such that each individual in $\mathcal{A}$ is in the extension of a concept $C_\mathcal{T}$ representing $\mathcal{T}$.

Definition 20 (P2RPQs)**.** A *positive 2-way regular path query* (P2RPQ) over a KB $\mathcal{K}$ is a formula $\exists \vec{x}.\varphi(\vec{x})$, where $\varphi(\vec{x})$ is built using $\wedge$ and $\vee$ from atoms of the form $C(z)$ and $R(z, z')$, with z, z' variables from $\vec{x}$ or individuals, C is an arbitrary concept, R is an arbitrary role, and where all atomic concepts and roles in φ occur in $\mathcal{K}$. △

Note that positive (regular path) queries naturally generalize unions of conjunctive (regular path) queries [43] by allowing for an unrestricted interaction of conjunction and disjunction[14], thus being in general also exponentially more compact.

Example 21. Consider the query q over a genealogy KB $\mathcal{K}$:

$$\begin{array}{c} \exists x, y, z.\ parent^* \cdot parent^{-*}(x, y) \wedge parent^-(x, z) \wedge parent^-(y, z) \\ \wedge male(x) \wedge \neg male(y) \wedge (\neg deity(x) \vee \neg deity(y)) \end{array}$$

Informally, q is true if there are relatives x and y that have a common child, z, and if not both of them are deities.

Let q be a P2RPQ, and let $\mathsf{varind}(q)$ denote the set of variables and individuals in q. Given an interpretation $\mathcal{I}$, let $\pi : \mathsf{varind}(q) \to \Delta^\mathcal{I}$ be a total function such that $\pi(a) = a^\mathcal{I}$ for each individual $a \in \mathsf{varind}(q)$.

We write $\mathcal{I}, \pi \models C(z)$ if $\pi(z) \in C^\mathcal{I}$, and $\mathcal{I}, \pi \models R(z, z')$ if $(\pi(z), \pi(z')) \in R^\mathcal{I}$. Let γ be the Boolean expression obtained from φ by replacing each atom α in φ with TRUE, if $\mathcal{I}, \pi \models \alpha$, and with FALSE otherwise. We say that π is a *match for* $\mathcal{I}$ *and* q, denoted $\mathcal{I}, \pi \models q$, if γ evaluates to TRUE. We

[14] Instead, no negation is allowed, whence the name.

say that $\mathcal{I}$ *satisfies* q, written $\mathcal{I} \models q$, if there is a match π for $\mathcal{I}$ and q. A KB $\mathcal{K}$ *entails* q, denoted $\mathcal{K} \models q$, if $\mathcal{I} \models q$ for each model $\mathcal{I}$ of $\mathcal{K}$.

Query entailment consists in verifying, given a KB $\mathcal{K}$ and a P2RPQ q, whether $\mathcal{K} \models q$. Note that, w.l.o.g., we consider here query entailment for Boolean queries, i.e., queries without free variables, since query answering for non-Boolean queries is polynomially reducible to query entailment.

6.2.2 Automata on Infinite Trees

Infinite trees are represented as prefix-closed (infinite) sets of words over $\mathbb{N}$ (the set of positive integers). Formally, an *infinite tree* is a set of words $T \subseteq \mathbb{N}^*$, such that if $x{\cdot}c \in T$, where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$. The elements of T are called *nodes*, the empty word ε is its *root*. For every $x \in T$, the nodes $x{\cdot}c$, with $c \in \mathbb{N}$, are the *successors* of x. By convention, $x{\cdot}0 = x$, and $x{\cdot}i{\cdot}{-1} = x$. The *branching degree* $d(x)$ of a node x is the number of its successors. If $d(x) \leq k$ for each node x of T, then T has *branching degree* k. An *infinite path* P of T is a prefix-closed set $P \subseteq T$ where for every $i \geq 0$ there exists a unique node $x \in P$ with $|x| = i$. A *labeled tree* over an alphabet Σ is a pair (T, V), where T is a tree and $V : T \to \Sigma$ maps each node of T to an element of Σ.

Let $\mathcal{B}(I)$ be the set of positive Boolean formulas built inductively from TRUE, FALSE, and atoms from a set I applying $\wedge$ and $\vee$. A set $J \subseteq I$ *satisfies* $\varphi \in \mathcal{B}(I)$, if assigning TRUE to the atoms in J and FALSE to those in $I \setminus J$ makes φ true.

A *two-way alternating tree automaton* (2ATA) running over infinite trees with branching degree k, is a tuple $\mathbf{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is the input alphabet; Q is a finite set of states; $\delta : Q \times \Sigma \to \mathcal{B}([k] \times Q)$, where $[k] = \{-1, 0, 1, \ldots, k\}$, is the transition function; $q_0 \in Q$ is the initial state; and F specifies the acceptance condition.

The transition function δ maps a state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a positive Boolean formula φ. Intuitively, each atom (c, q') in φ corresponds to a new copy of the automaton going in the direction given by c and starting in state q'. E.g., let $k = 2$ and $\delta(q_1, \sigma) = (1, q_2) \wedge (1, q_3) \vee (-1, q_1) \wedge (0, q_3)$. If $\mathbf{A}$ is in the state q_1 and reads the node x labeled with σ, it proceeds by sending off either two copies, in the states q_2 and q_3 respectively, to the first successor of x (i.e., $x{\cdot}1$), or one copy in the state q_1 to the predecessor of x (i.e., $x{\cdot}{-1}$) and one copy in the state q_3 to x itself (i.e., $x{\cdot}0$).

Informally, a run of a 2ATA $\mathbf{A}$ over a labeled tree (T, V) is a labeled tree (T_r, r) in which each node n is labeled by an element $r(n) = (x, q) \in T \times Q$ and describes a copy of $\mathbf{A}$ that is in the state q and reads the node x of T; the labels of adjacent nodes must satisfy the transition function of $\mathbf{A}$. Formally, a *run* (T_r, r) is a $T{\times}Q$-labeled tree satisfying:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Let $y \in T_r$, with $r(y) = (x, q)$ and $\delta(q, V(x)) = \varphi$. Then there is a set $S = \{(c_1, q_1), \ldots, (c_h, q_h)\} \subseteq [k] \times Q$ s.t.
   - S satisfies φ and
   - for all $1 \leq i \leq h$, we have that $y{\cdot}i \in T_r$, $x{\cdot}c_i$ is defined, and $r(y{\cdot}i) = (x{\cdot}c_i, q_i)$.

A run (T_r, r) is *accepting*, if it satisfies the *parity* condition that for every infinite path π, there is an *even* i such that $\mathsf{Inf}(\pi) \cap G_i \neq \emptyset$ and $\mathsf{Inf}(\pi) \cap G_{i-1} = \emptyset$, where $F = (G_1, \ldots, G_m)$ is a finite sequence of sets of states with $G_1 \subseteq \cdots \subseteq G_m = Q$, and $\mathsf{Inf}(\pi) \subseteq Q$ denotes the states that occur infinitely often in π (as second components of node labels). The *nonemptiness problem* for 2ATAs is deciding whether the set $\mathcal{L}(\mathbf{A})$ of trees accepted by a given 2ATA $\mathbf{A}$ is nonempty. We make use of the following result.

Theorem 22. [149] *For any 2ATA* $\mathbf{A}$ *with* n *states, parity condition of length* m*, and input alphabet with* ℓ *elements, nonemptiness of* $\mathbf{A}$ *is decidable in time exponential in* n *and polynomial in* m *and* ℓ*. There is*

a one-way nondeterministic tree automaton (1NTA) $\mathbf{A}_1$ *with* $2^{O(n)}$ *states and parity condition of length* $O(m)$ *such that* $\mathcal{L}(\mathbf{A}) = \mathcal{L}(\mathbf{A}_1)$.

6.3 Deciding KB Satisfiability via Automata

For many DLs including $\mathcal{ALCQIb}_{reg}$, the standard reasoning tasks are naturally solvable by tree-automata, thanks to their *tree model property*: each satisfiable concept C has a tree-shaped model. This is similar in the presence of a TBox. For an ABox $\mathcal{A}$ this fails, since the assertions in $\mathcal{A}$ may arbitrarily connect individuals. While a satisfiable $\mathcal{ALCQIb}_{reg}$ KB $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ may lack a tree-shaped model, it always has a forest-shaped *canonical model*, in which each individual is the root of a tree-shaped model of $\mathcal{T}$. This property is usually sufficient to adapt algorithms for concept satisfiability to decide KB satisfiability. In particular, automata-based algorithms have been adapted using the *precompletion* technique [143], in which after a reasoning step on the ABox, automata are used to verify the existence of a tree-shaped model rooted at each ABox individual.

Our approach is different. We represent forest-shaped interpretations as trees $\mathbf{T}$, and encode $\mathcal{K}$ into an automaton $\mathbf{A}_\mathcal{K}$ that accepts $\mathbf{T}$ iff $\mathbf{T}$ corresponds to a canonical model of $\mathcal{K}$. To the best of our knowledge, this is the first algorithm that deals with ABox assertions and individuals directly in the automaton. This enables us to extend the automata-based algorithm also to query answering.

We denote by $CL(C_\mathcal{T})$ the *(syntactic) closure* of $C_\mathcal{T}$ as defined in [40]. Intuitively, it contains all the concepts and roles that may occur when $C_\mathcal{T}$ is decomposed during a run of an automaton on a tree representing a model of $\mathcal{K}$. It contains $C_\mathcal{T}$ and it is closed under subconcepts and their negations. It also contains some basic roles (with their corresponding subroles and negations), and some concepts that may occur when decomposing a subconcept of $C_\mathcal{T}$ in which complex concepts occur (e.g., if $\exists(R \circ R').C \in CL(C_\mathcal{T})$ then $\exists R.\exists R'.C \in CL(C_\mathcal{T})$). We assume that $CL(C_\mathcal{T})$ also contains a_i and $\neg a_i$ for each ABox individual a_i, plus d and $\neg d$, where d is a new *dummy* symbol. Note that $|CL(C_\mathcal{T})|$ is linear in the length of $\mathcal{K}$. Sometimes we consider expressions E in *negation normal form*, denoted $\mathsf{nnf}E$, in which negations are pushed inside as much as possible. We let $CL^{nnf}(C_\mathcal{T}) = \{\mathsf{nnf}E \mid E \in CL(C_\mathcal{T})\}$.

Every satisfiable $\mathcal{ALCQIb}_{reg}$ concept $C_\mathcal{T}$ has a tree-model with branching degree $k_{C_\mathcal{T}} = O(|CL(C_\mathcal{T})|)$ [40, 143].

Satisfiable $\mathcal{ALCQIb}_{reg}$ KBs have a weaker property:

Theorem 23. *Every satisfiable* $\mathcal{ALCQIb}_{reg}$ *KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *has a canonical model* $\mathcal{I}$ *that comprises a set of tree-shaped models of* $C_\mathcal{T}$ *with branching degree* $k_{C_\mathcal{T}}$*, whose roots are the individuals in* $\mathcal{A}$ *(which might be interconnected).*

We represent such a canonical model $\mathcal{I}$ as a tree $\mathbf{T}_\mathcal{I}$. Let $\mathcal{J}_\mathcal{K} = \{a_1, \ldots, a_m\}$, and let $S = \{t_1, \ldots, t_n\}$ be the set of tree-shaped models of $C_\mathcal{T}$ in $\mathcal{I}$ (each with branching degree $k_{C_\mathcal{T}}$). As in [40], we represent each such t_j as a labeled tree. Each node x is labeled with a set σ that contains the atomic concepts that are true in x, and the basic roles that connect the predecessor of x to x. The label of the root of t_j also contains the names of the individuals in $\mathcal{J}_\mathcal{K}$ which it interprets, but no basic roles. The root of $\mathbf{T}_\mathcal{I}$ is a new node whose children are the roots of all trees t_j. Its label is $\{r\} \cup \{Pij \mid \langle a_i^\mathcal{I}, a_j^\mathcal{I} \rangle \in P^\mathcal{I}\}$. Since each t_j is rooted at some a_i, we have $n \leq m$. If $n < m$, the root has $m - n$ dummy children labeled d. Note that the branching degree is $|\mathcal{J}_\mathcal{K}|$ at the root and $k_{C_\mathcal{T}}$ at all other levels.

We now construct from $\mathcal{K}$ a 2ATA $\mathbf{A}_\mathcal{K}$ that accepts a given tree $\mathbf{T}$ iff $\mathbf{T} = \mathbf{T}_\mathcal{I}$ for some canonical model $\mathcal{I}$ of $\mathcal{K}$. [40] presented an automaton $\mathbf{A}_\mathcal{K} = (\Sigma_\mathcal{K}, S, \delta, s_0, F)$ for deciding concept satisfiability in $\mathcal{ALCQIb}_{reg}$. We adapt and expand $\mathbf{A}_\mathcal{K}$ to handle the ABox. The alphabet is $\Sigma_\mathcal{K} = 2^{\mathcal{C}_\mathcal{K} \cup \mathcal{R}_\mathcal{K} \cup \mathcal{J}_\mathcal{K} \cup \{r\} \cup \{d\} \cup PI}$, where $PI = \{Pij \mid a_i, a_j \in \mathcal{J}_\mathcal{K} \text{ and } P \in \mathcal{R}_\mathcal{K}\}$; the set of states is $S = \{s_0\} \cup CL^{nnf}(C_\mathcal{T}) \cup S_\mathcal{A} \cup S_Q$ where s_0 is the initial state. The acceptance condition is

$F = (\emptyset, \{\forall R^*.C \in CL^{nnf}(C_\mathcal{T})\})$ (concepts $\exists R^*.C$ are not included in the acceptance condition, and are satisfied in all accepting runs, see [40]).[15]

Intuitively, when $\mathbf{A}_\mathcal{K}$ is in a state $s \in CL^{nnf}(C_\mathcal{T})$ and visits a node x of the tree, it must check that s holds in x. The set $S_\mathcal{A}$ contains states of the form Qij to verify whether ABox individuals a_i and a_j are related by a role Q, and states of the form $\langle j, \exists Q.C\rangle$ and $\langle j, \forall Q.C\rangle$ to check whether a_j satisfies a concept of the form $\exists Q.C$ and $\forall Q.C$. The set S_Q contains states of the form $\langle \geqslant n\, Q.C, i, j\rangle$ and $\langle k, \geqslant nQC, i, j\rangle$ for $\geqslant \in \{\geqslant, \leqslant\}$, which check the number restrictions. Intuitively, i stores how many successors of the current node have been navigated, and j how many of them are reached through Q and labeled with C. Similarly, the states $\langle k, \geqslant nQC, i, j\rangle$ are used to verify that an individual a_k satisfies the concept $\geqslant nQC$.

The transition function δ is as follows. First, for each $\sigma \in \Sigma_\mathcal{K}$ with $r \in \sigma$ we define $\delta(s_0, \sigma) = F_1 \wedge \cdots \wedge F_7$ from the initial state s_0, which verifies that the root contains r; that the level one nodes properly represent the individuals in the ABox (F_1–F_3); that all ABox assertions are satisfied (F_4–F_6); and that every non-dummy node at level one is the root of a tree representing a model of $C_\mathcal{T}$ (F_7):

$$
\begin{aligned}
F_1 &= \textstyle\bigwedge_{1\leq i\leq|\mathcal{J}_\mathcal{K}|}((\bigvee_{1\leq j\leq|\mathcal{J}_\mathcal{K}|}(i,a_j)\wedge(i,\neg d))\vee(i,d))\\
F_2 &= \textstyle\bigwedge_{1\leq i\leq|\mathcal{J}_\mathcal{K}|}\bigvee_{1\leq j\leq|\mathcal{J}_\mathcal{K}|}(j,a_i)\\
F_3 &= \textstyle\bigwedge_{1\leq i<j\leq|\mathcal{J}_\mathcal{K}|}(\bigwedge_{1\leq k\leq|\mathcal{J}_\mathcal{K}|}(i,\neg a_k)\vee(j,\neg a_k))\\
F_4 &= \textstyle\bigwedge_{a_i\neq a_j\in\mathcal{A}}(\bigwedge_{1\leq k\leq|\mathcal{J}_\mathcal{K}|}(k,\neg a_i)\vee(k,\neg a_j))\\
F_5 &= \textstyle\bigwedge_{A(a_j)\in\mathcal{A}}(\bigvee_{1\leq i\leq|\mathcal{J}_\mathcal{K}|}(i,a_j)\wedge(i,A))\\
F_6 &= \textstyle\bigwedge_{P(a_i,a_j)\in\mathcal{A}}(0,Pij)\\
F_7 &= \textstyle\bigwedge_{1\leq i\leq|\mathcal{J}_\mathcal{K}|}((i,\mathsf{nnf}C_\mathcal{T})\vee(i,d))
\end{aligned}
$$

Additional transitions ensure that r and each a_i do not occur anywhere else in the tree. Then, for each concept in $CL^{nnf}(C_\mathcal{T})$ and each $\sigma \in \Sigma_\mathcal{K}$, there are transitions that recursively decompose concepts and roles, and move to appropriate states of the automaton and nodes of the tree.

Concepts $\forall R^*.C$ and $\exists R^*.C$ are propagated using the equivalent concepts $C \sqcap \forall R.\forall R^*.C$ and $C \sqcup \exists R.\exists R^*.C$, respectively. Most of these transitions are as in [40]. To verify that a concept of the form $\forall Q.C$, $\exists Q.C$, $\geqslant n\, Q.C$ or $\leqslant n\, Q.C$ is satisfied by a node x, all the nodes that reach or are reachable from x must be navigated. We need different transitions for a node x (*i*) at level one and (*ii*) at all other levels. In case (*ii*), the predecessor and the successors of x are navigated as usual. In case (*i*), the transitions must consider the other individual nodes that are connected to x via some role, which are stored in the root label. Therefore, the transitions must send suitable copies of the automaton to navigate the successors, and send a copy of the automaton up to the root. As an example, we provide the transitions for the quantifiers; the number restrictions are handled similarly. If $\sigma \cap (\mathcal{J}_\mathcal{K} \cup \{d\}) \neq \emptyset$, we have transitions:

$$\delta(\exists Q.C, \sigma) = \textstyle\bigvee_{a_j\in\sigma}(-1, \langle j, \exists Q.C\rangle) \vee \bigvee_{1\leq i\leq k_{C_\mathcal{T}}}((i,Q)\wedge(i,C))$$

$$
\begin{aligned}
\delta(\forall Q.C, \sigma) = &\textstyle\bigwedge_{a_j\in\sigma}(-1, \langle j, \forall Q.C\rangle) \wedge\\
&\textstyle\bigwedge_{1\leq i\leq k_{C_\mathcal{T}}}((i,\mathsf{nnf}Q)\vee(i,C))
\end{aligned}
$$

Further, for each $\sigma \in \Sigma_\mathcal{K}$ and $\langle j, \exists Q.C\rangle$, $\langle j, \forall Q.C\rangle$ in $S_\mathcal{A}$,

[15] We could also use a *Büchi condition* $\{\forall R^*.C \in CL^{nnf}(C_\mathcal{T})\}$.

$$\delta(\langle j, \exists Q.C\rangle, \sigma) = \bigvee_{0\le i\le|\mathcal{J}_\mathcal{K}|} \Big(\bigvee_{0\le k\le|\mathcal{J}_\mathcal{K}|} ((0, Qjk) \wedge (i, a_k) \wedge (i, C))\Big)$$
$$\delta(\langle j, \forall Q.C\rangle, \sigma) = \bigwedge_{0\le i\le|\mathcal{J}_\mathcal{K}|} \Big(\bigwedge_{0\le k\le|\mathcal{J}_\mathcal{K}|} ((0, \mathsf{nnf}Qjk) \vee (i, \neg a_k) \vee (i, C))\Big)$$

Concepts and roles are recursively decomposed. When reaching the atomic level, it is checked whether the node label σ contains the corresponding atomic symbol. Thus, for each $s \in \mathcal{C}_\mathcal{K} \cup \mathcal{R}_\mathcal{K} \cup \mathcal{J}_\mathcal{K} \cup d$:

$$\delta(s, \sigma) = \begin{cases} \mathsf{TRUE}, & \text{if } s \in \sigma \\ \mathsf{FALSE}, & \text{if } s \notin \sigma \end{cases} \qquad \delta(\neg s, \sigma) = \begin{cases} \mathsf{TRUE}, & \text{if } s \notin \sigma \\ \mathsf{FALSE}, & \text{if } s \in \sigma \end{cases}$$

Further transitions verify whether ABox individuals are connected via some atomic role by checking the label of the root. For each $\sigma \in \Sigma_\mathcal{K}$ and $Pij \in S_\mathcal{A}$ with $P \in \mathcal{R}_\mathcal{K}$:

$$\delta(Pij, \sigma) = \begin{cases} \mathsf{TRUE}, & \text{if } (Pij \in \sigma) \text{ or } (P^- ji \in \sigma) \\ \mathsf{FALSE}, & \text{otherwise} \end{cases}$$

A run of $\mathbf{A}_\mathcal{K}$ on an infinite tree $\mathbf{T}$ starts at the root, and moves to each individual node to check that $C_\mathcal{T}$ holds there. To this end, $\mathsf{nnf}C_\mathcal{T}$ is recursively decomposed while appropriately navigating the tree, until $\mathbf{A}_\mathcal{K}$ arrives at atomic elements, which are checked locally.

Given a labeled tree $\mathbf{T} = (T, V)$ accepted by $\mathbf{A}_\mathcal{K}$, we define an interpretation $\mathcal{I}_\mathbf{T}$ for $\mathcal{K}$. The domain $\Delta^{\mathcal{I}_\mathbf{T}}$ is given by the nodes x in $\mathbf{T}$ with $a_i \in V(x)$ for some individual a_i, and the nodes in $\mathbf{T}$ that are reachable from any such x through the roles. The extensions of concepts and roles are determined by the labels of the nodes in $\mathbf{T}$.

Lemma 24. *Let $\mathbf{T}$ be a labeled tree accepted by $\mathbf{A}_\mathcal{K}$. Then $\mathcal{I}_\mathbf{T}$ is a model of $\mathcal{K}$.*

Conversely, given a canonical model $\mathcal{I}$ of $\mathcal{K}$, we can construct from it a labeled tree $\mathbf{T}_\mathcal{I}$ that is accepted by $\mathbf{A}_\mathcal{K}$.

Lemma 25. *$\mathbf{A}_\mathcal{K}$ accepts $\mathbf{T}_\mathcal{I}$ for each canonical model $\mathcal{I}$ of $\mathcal{K}$.*

From Lemmas 24 and 25 and Theorem 23, we get:

Theorem 26. *An $\mathcal{ALCQI}b_{reg}$ KB $\mathcal{K}$ is satisfiable iff the set of trees accepted by $\mathbf{A}_\mathcal{K}$ is nonempty.*

Under unary encoding of numbers in restrictions, the number of states of $\mathbf{A}_\mathcal{K}$ is polynomial in the size of $\mathcal{K}$. Since $\Sigma_\mathcal{K}$ is single exponential in the size of $\mathcal{K}$, by Theorems 22 and 26 we get an optimal upper bound for KB satisfiability (a matching lower bound holds already for much weaker DLs [13]).

Corollary 27. *For $\mathcal{ALCQI}b_{reg}$, KB satisfiability is* EXPTIME*-complete.*

6.4 Query Answering via Automata

We address now the problem of entailment of P2RPQs in $\mathcal{ALCQI}b_{reg}$ KBs. Consider a (Boolean) P2RPQ q over a KB $\mathcal{K}$. We first show that, in order to check whether $\mathcal{K} \models q$, it is sufficient to restrict attention to canonical models. This is a consequence of the possibility to unravel an arbitrary counterexample model for entailment into a canonical model (cf. Theorem 23), and the fact that since the query does not contain negative information, there will still be no match for the unraveled model and the query.

Lemma 28. *$\mathcal{K} \not\models q$ if and only if there is a canonical model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models q$.*

This result allows us to exploit tree-automata based techniques also for query entailment. Specifically, we consider trees representing canonical models over an alphabet extended with additional atomic concepts, one for each variable in q, each of which is satisfied in a single node of the tree. The intuition behind the use of such trees is that, since the existentially quantified "variables" appear explicitly in the tree, a 2ATA $\mathbf{A}_q$ can easily check the existence of a match for (the interpretation corresponding to) the tree and q. We show now how to construct such a 2ATA.

Let $q = \exists \vec{x}.\varphi(\vec{x})$ be a P2RPQ over $\mathcal{K}$, and let *atomsq* be the set of atoms appearing in q. We consider $\mathcal{C}_\mathcal{K}$ to be enriched by the set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of variables in $\vec{x}$, and additionally may make use of the ABox individuals in $\mathcal{J}_\mathcal{K}$ in place of atomic concepts.[16]

Let then $U = (\bigcup_{P \in \mathcal{R}}(P \cup P^-))^*$. For each $\alpha \in$ *atomsq*, we define

$$C_\alpha = \begin{cases} \exists U.(C \sqcap z), & \text{if } \alpha = C(z) \\ \exists U.(z_1 \sqcap \exists R.z_2), & \text{if } \alpha = R(z_1, z_2) \end{cases}$$

where $z, z_1, z_2 \in \mathcal{J}_\mathcal{K} \cup \mathcal{X}$.

We define the 2ATA $\mathbf{A}_q = (\Sigma_q, S_q, \delta, s_0, F)$ as follows

- $\Sigma_q = \Sigma_\mathcal{K} \cup \mathcal{X}$;
- S_q is defined similarly as for $\mathbf{A}_\mathcal{K}$, except that we use $\bigcup_{\alpha \in \mathit{atomsq}} CL^{nnf}(C_\alpha)$ instead of $CL^{nnf}(C_\mathcal{T})$.
- The transitions from the initial state are defined for all labels σ containing the symbol r (identifying the root node) as $\delta(s_0, \sigma) = F_1 \wedge F_2 \wedge F_3 \wedge F_q \wedge F_v$, where:
  - F_1, F_2, and F_3 are as for $\mathbf{A}_\mathcal{K}$;
  - F_q is obtained from $\varphi(\vec{x})$ by replacing each atom α with $(0, C_\alpha)$ (and by considering $\wedge$ and $\vee$ as the analogous connectives in a 2ATA transition);
  - F_v checks that each atomic concept $x \in \mathcal{X}$ appears exactly once in the tree (this requires new states in $\mathbf{A}_q$);
- F is defined as for $\mathbf{A}_\mathcal{K}$.

When $\mathbf{A}_q$ is in a state C_α in the root node (the only node labeled r), it does not "decompose" C_α as usual. Instead, it checks that the concept C_α is satisfied starting from a node at level one representing ABox individuals. This is done by the following transitions, one for each $\alpha \in$ *atomsq* and σ containing α:

$$\delta(C_\alpha, \sigma) = \bigvee_{1 \leq i \leq |\mathcal{J}_\mathcal{K}|}(i, C_\alpha)$$

Then, $\mathbf{A}_q$ contains transitions analogous to those of $\mathbf{A}_\mathcal{K}$ to check that the various concepts C_α are satisfied. Exploiting that the concepts representing variables are enforced to be satisfied in a single node of the tree, and that under this assumption the concepts C_α correctly represent the atoms of q, we can show the following result.

Lemma 29. *Let $\mathcal{I}_\mathbf{T}$ be the canonical interpretation defined from a tree $\mathbf{T}$ as above. Then $\mathbf{A}_q$ accepts $\mathbf{T}$ iff there is a match for $\mathcal{I}_\mathbf{T}$ and q in which each variable x of q is mapped to the (only) object that is an instance of concept x.*

[16] We do not need to enrich the alphabet of atomic concepts by the ABox individuals though, since they are already in it.

We then convert $\mathbf{A}_q$ into an equivalent 1NTA $\mathbf{A}_q^1$. By Theorem 22, the number of states (resp., parity condition) of $\mathbf{A}_q^1$ is exponential (resp., polynomial) in the number of states of $\mathbf{A}_q$, i.e., in the sum of the size of q and $\mathcal{K}$.

We project out variables from $\mathbf{A}_q^1$ obtaining a 1NTA $\mathbf{A}_q^2$ of size not larger than that of $\mathbf{A}_q^1$. By construction, since $\mathbf{A}_q^2$ is a one-way automaton and it has been obtained from $\mathbf{A}_q^1$ by projecting away the variable symbols $\mathcal{X}$, we have that a tree $\mathbf{T}$ is accepted by $\mathbf{A}_q^2$ iff there is a match for $\mathcal{I}_\mathbf{T}$ and q.

We complement $\mathbf{A}_q^2$, obtaining a 1NTA $\mathbf{A}_{\neg q}$. The number of states (resp., acceptance condition) of $\mathbf{A}_{\neg q}$ is exponential (resp., polynomial) in the number of states of $\mathbf{A}_q^2$ [92], i.e., double exponential (resp., polynomial) in the size of q and $\mathcal{K}$. We have that a tree $\mathbf{T}$ is accepted by $\mathbf{A}_{\neg q}$ iff there is no match for $\mathcal{I}_\mathbf{T}$ and q.

We construct a 1NTA $\mathbf{A}_{\mathcal{K} \not\models q}$ that accepts the intersection of the languages accepted by $\mathbf{A}_\mathcal{K}$ and $\mathbf{A}_{\neg q}$. This can be done by first converting $\mathbf{A}_\mathcal{K}$ to a 1NTA whose number of states (resp., acceptance condition) is exponential (resp., linear) in the size of $\mathcal{K}$, and then constructing the product automaton with $\mathbf{A}_{\neg q}$. The number of states (resp., acceptance condition) of $\mathbf{A}_{\mathcal{K} \not\models q}$ is still double exponential (resp., polynomial) in the size of q and $\mathcal{K}$.[17]

Since a tree $\mathbf{T}$ is accepted by $\mathbf{A}_\mathcal{K}$ iff $\mathcal{I}_\mathbf{T}$ is a canonical model of $\mathcal{K}$, while it is accepted by $\mathbf{A}_{\neg q}$ iff there is no match for $\mathcal{I}_\mathbf{T}$ and q, every tree accepted by $\mathbf{A}_{\mathcal{K} \not\models q}$ represents a counterexample to $\mathcal{K} \models q$. On the other hand, if a tree $\mathbf{T}$ is not accepted by $\mathbf{A}_{\mathcal{K} \not\models q}$, then either it is not accepted by $\mathbf{A}_\mathcal{K}$, in which case $\mathcal{I}_\mathbf{T}$ is not a model of $\mathcal{K}$, or it is not accepted by $\mathbf{A}_{\neg q}$, in which case it is accepted by $\mathbf{A}_q^2$, i.e., there is a match for $\mathcal{I}_\mathbf{T}$ and q. Hence the tree does not represent a counterexample to $\mathcal{K} \models q$. As a consequence, we get:

Lemma 30. *There exists a canonical counterexample to $\mathcal{K} \models q$ iff the set of trees accepted by $\mathbf{A}_{\mathcal{K} \not\models q}$ is not empty.*

By Lemma 28, and the fact that non-emptiness of 1NTAs can be decided in time linear in the number of states of the automaton and exponential in the acceptance condition, see [149], we get the following result.

Theorem 31. *For every $\mathcal{ALCQI}b_{reg}$ knowledge base $\mathcal{K}$ and P2RPQ query q, we have that $\mathcal{K} \models q$ iff the set of trees accepted by $\mathbf{A}_{\mathcal{K} \not\models q}$ is not empty. Moreover, $\mathcal{K} \models q$ is decidable in double exponential time in the size of q and the number of atomic concepts, roles, and individuals in $\mathcal{K}$ and single exponential in the size of $\mathcal{K}$.*

Our results apply also to $\mathcal{SHIQ}$. Given a $\mathcal{SHIQ}$ KB $\mathcal{K}$, it can be rewritten as an $\mathcal{ALCQI}b_{reg}$ KB $\mathcal{K}'$ expressing the role hierarchy with role conjunction (complex roles are not allowed in the ABox, thus it must be closed w.r.t. the hierarchy) and propagating value restrictions over transitive roles by means of TBox axioms [143]. Although this reduction does not preserve query entailment, the models of $\mathcal{K}$ and $\mathcal{K}'$ differ only in the interpretation of transitive roles. For a query q, deciding $\mathcal{K} \models q$ can be reduced to deciding $\mathcal{K}' \models q'$, where q' is obtained from q by replacing every transitive role R in q with $R \circ R^*$.

6.5 EXPSPACE-Hardness of Query Answering

In this section, we provide a lower bound on answering PRPQs (i.e., P2RPQs without inverses) over $\mathcal{ALC}$ KBs.

Theorem 32. *Given a PRPQ q and a $\mathcal{ALC}$ knowledge base $\mathcal{K}$, deciding whether $\mathcal{K} \models q$ is* EXPSPACE*-hard.*

[17] Note that, if only atomic concepts and (regular expressions over) atomic roles are used in q, then the number of states of $\mathbf{A}_{\mathcal{K} \not\models q}$ is single exponential in the size of $\mathcal{K}$.

The proof is by a reduction from tiling problems, inspired by a similar reduction to query containment over semi-structured data [43].

A tiling problem consists of a finite set Δ of tile types, two binary relations H and V over Δ, representing horizontal and vertical adjacency relations, respectively, and two distinguished tile types $t_S, t_F \in \Delta$. Deciding whether for a given a number n in unary, a region of the integer plane of size $2^n \times k$, for some k, can be tiled consistently with H and V, such that the left bottom tile of the region has type t_S and the right upper tile has type t_F, can be shown to be EXPSPACE-complete [146].

We construct an $\mathcal{ALC}$ KB $\mathcal{K}$ and a query q such that $\mathcal{K} \models q$ iff there is no correct tiling, as follows. A tiling is spanned row by row by a sequence of objects. Each object represents one tile and is connected by a specific role to the next tile. For the connections, we use the following two roles:

- N connecting tiles within the same row;
- L connecting the last tile of row i to the first of row $i+1$.

The properties (i.e., the atomic concepts) attached to an object are the n bits $B_1, \ldots, B_n$ of a counter for its address within the row, and its type. For that, we use pairwise disjoint concepts $D_1, \ldots, D_k$, where $\Delta = \{t_1, \ldots, t_k\}$.

We encode in $\mathcal{K}$ the following two conditions:

1. The first ensures that the counters progress correctly. It consists of $O(n)$ standard axioms involving $B_1, \ldots, B_n$ and N, which encode a counter bit by bit. Further axioms ensure that, if at least one bit is 0, there is an N successor but no L successor, and reset the counter otherwise.

$$\begin{aligned} \neg B_1 \sqcup \cdots \sqcup \neg B_n &\sqsubseteq \exists N.\top \sqcap \forall L.\bot \\ B_1 \sqcap \cdots \sqcap B_n &\sqsubseteq \exists L.(\neg B_1 \sqcap \cdots \sqcap \neg B_n) \sqcap \forall N.\bot \end{aligned}$$

2. The second ensures that there are no errors w.r.t. the horizontal adjacency relation H. For each tile type D_i,

$$D_i \sqsubseteq \bigsqcup_{(D_i, D_j) \in H} (\forall N.D_j \sqcap \forall L.D_j).$$

The query q checks the failure of the vertical adjacency V on the candidate tilings given by the models of $\mathcal{K}$. It asks whether two objects exist at distance 2^n (i.e., representing vertically adjacent tiles) with an error according to V. That the objects are exactly 2^n steps apart is achieved by ensuring that they have the same n bits and are connected by a (possibly void) sequence of N-steps, followed by one L-step, and by a (possibly void) sequence of N-steps. We have

$$\begin{aligned} q &= \exists x, y.\, \mathit{Vert} \wedge \mathit{Err} \wedge G_1 \wedge \cdots \wedge G_n, \quad \text{where} \\ \mathit{Vert} &= (N^* \circ L \circ N^*)(x, y), \\ \mathit{Err} &= \bigvee_{(D_i, D_j) \notin V} (D_i(x) \wedge D_j(y)), \\ G_i &= (B_i(x) \wedge B_i(y)) \vee (\neg B_i(x) \wedge \neg B_i(y)), \text{ for } 1 \leq i \leq n. \end{aligned}$$

The complete KB $\mathcal{K}$ entails q iff there is no correct tiling. Note that only Vert uses a regular expression. If we have transitive roles and role hierarchies, we can replace it in q by

$$\begin{aligned} \mathit{Vert}' = &(N_t(x, z_1) \wedge L(z_1, z_2) \wedge N_t(z_2, y)) \vee \\ &(N_t(x, z_1) \wedge L(z_1, y)) \vee (L(x, z_2) \wedge N_t(z_2, y)) \end{aligned}$$

where N_t is a transitive super-role of N, and z_1 and z_2 are existentially quantified variables. This shows that answering positive (existential) queries without regular expressions over KBs in $\mathcal{ALC}$ plus transitive roles and role hierarchies, and hence in $\mathcal{SH}$, is EXPSPACE-hard.

Finally, using an encoding closer to [43] where each tile is a block of $n+1$ objects, and the bits and tile types are encoded by roles, one can show that answering conjunctive regular path queries over KBs which only use existential roles and disjunction is EXPSPACE-hard.

6.6 Conclusion

In this paper, we have substantially pushed the frontier of decidable query answering over expressive DLs, which is an active area of research driven by the growing interest to deploy DLs to various application areas related to AI. As we have shown, the rich class of positive two-way regular path queries (P2RPQs) is decidable for $\mathcal{ALCQI}b_{reg}$ KBs by means of automata-techniques; on the other hand, query answering has an EXPSPACE-lower bound already in settings where one of $\mathcal{K}$ and Q is rather plain.

Recent results show that the 2EXPTIME upper bound we provide in this paper is indeed tight [105]. However, such a hardness result makes essential use of inverse roles, and the precise complexity of PRPQs remains open. Finally, it would be interesting to see how far automata-based techniques similar to the ones in this paper can be utilized to push the decidability frontier of query answering in expressive DLs, both on the side of the query and the KB.

7 Unrestricted Conjunctive Queries Data-Oriented Description Logics

One of the most important research directions in Description Logics (DLs) is concerned with the trade-off between expressive power and computational complexity of sound and complete reasoning. Research carried out in the past on this topic has shown that many DLs with efficient, i.e., worst-case polynomial time, reasoning algorithms lack modeling power required in capturing conceptual models and basic ontology languages, while most DLs with sufficient modeling power suffer from inherently worst-case exponential time behavior of reasoning [8, 24].

Although the requirement of polynomially tractable reasoning might be less stringent when dealing with relatively small ontologies, we believe that the need for efficient reasoning algorithms is of paramount importance when the ontology system is to manage large amount of objects (e.g., from thousands to millions of instances). This is the case of several important applications where the use of ontologies is advocated nowadays. For example, in the Semantic Web, ontologies are often used to describe the relevant concepts of Web repositories, and such repositories may incorporate very large data sets, which constitute the instances of the concepts in the ontology. In such cases, two requirements emerge that are typically overlooked in DLs. First, the number of objects in the knowledge base requires managing instances of concepts (i.e., ABoxes) in secondary storage. Second, significant queries to be posed to the knowledge base are more complex than the simple queries (i.e., concepts and roles) usually considered in DL research. Another interesting context where these requirements are relevant is data integration, where an ontology is used as a conceptual, unified view of data stored in a collection of heterogeneous sources. Again, queries posed to this unified representation are usually more complex than just simple concept and role expressions, and the amount of data accessed through the ontology is likely to be of significant size. Unfortunately, in these contexts, whenever the complexity of reasoning is exponential in the number of instances (as for example in Fact[18], RACER[19], and in [39]), there is little hope for effective instance management and query answering algorithms.

In this section we deal with *reasoning* in the *DL-Lite* family, a family of DLs specifically tailored to capture basic ontology languages, while keeping all reasoning tasks tractable, in particular, with polynomial time complexity with respect to the size of the knowledge base. Notably, reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries, i.e., unions of conjunctive queries, over the set of instances maintained in secondary storage.

Our specific contributions are the following:

1. We define the *DL-Lite* family, and show that the DLs of this family are rich enough to capture significant ontology languages. As usual, a knowledge base expressed in any logic of the *DL-Lite* family is constituted by a TBox and an ABox, where the first component specifies general properties of concepts and roles, whereas the second component specifies the instances of concepts and roles. The basic logic of the family is called *DL-Lite*$_{core}$, and allows for expressing (cyclic) ISA assertions on concepts, disjointness between concepts, role-typing, participation constraints, i.e., assertions stating that all instances of a concept participate to a specified role, and non-participation constraints. The two other logics studied in this section are called *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, respectively. The former adds to the core the possibility of expressing functionality restrictions on roles, whereas the latter adds ISA and disjointness assertions between roles. Although at first sight both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ appear to be very simple DLs, the kind of modeling constructs in these logics makes them suitable for expressing a variety of representation languages widely adopted in different contexts, such as basic ontology languages, conceptual data

[18] http://www.cs.man.ac.uk/~horrocks/FaCT/

[19] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

models (e.g., Entity-Relationship [46]), and object-oriented formalisms (e.g., basic UML class diagrams[20]).

2. For all DLs of the *DL-Lite* family, we present techniques for the usual reasoning tasks of DLs, such as concept and role subsumption, instance checking, and knowledge base satisfiability. We show that all these tasks are computationally tractable. More precisely, the time complexity of both concept and role subsumption is polynomial in the size of the knowledge base, whereas both knowledge base satisfiability and instance checking are in polynomial time with respect to the size of the knowledge base, and in LOGSPACE with respect to the size of the ABox only, i.e., in what we can call data complexity [148]. We call description logics with LOGSPACE data complexity *data-oriented description logics*.

3. For all DLs of the *DL-Lite* family, we present algorithms for answering unions of conjunctive queries posed to knowledge bases expressed in such DLs, and we show that their complexity is polynomial with respect to the size of the whole knowledge base, and in LOGSPACE with respect to the size of the ABox only. Note that this is one of the few results on answering complex queries (i.e., not corresponding simply to a concept or a role) over a DL knowledge base [39]. In fact, answering conjunctive queries in DLs is a challenging problem, even in the case of *DL-Lite*$_{core}$, where the combination of constructs expressible in the knowledge base does not pose particular difficulties to TBox reasoning. Indeed, in spite of the simplicity of *DL-Lite*$_{core}$ TBoxes, the ability of taking TBox knowledge into account during the process of answering (unions of) conjunctive queries goes beyond the two-variable fragments (with counting) of first-order logic represented by DLs [8]. Finally, we observe that the worst-case complexity of query answering is exponential in the size of the queries, but this is unavoidable, as the complexity of database query evaluation is already exponential. Overall, our complexity results show that, despite the expressive power of the DLs of the *DL-Lite* family, the complexity of query answering in these logics is no worse than traditional query evaluation in relational databases.

A notable consequence of the results presented in this report is that our approach is perfectly suited to representing ABox assertions as relations managed in secondary storage by a Data Base Management System (DBMS). Indeed, our query answering algorithms are based on the idea of expanding the original query into a set of queries that can be directly evaluated by an SQL engine over the ABox, thus taking advantage of well-established query optimization strategies supported by current industrial strength relational technology. Note that this was one of the motivations behind several research works done on CLASSIC in the 80's [25].

We finally point that from the results on the data complexity of DLs given in the TONES Deliverable D10 [29], it follows in particular that the DLs of the *DL-Lite* family are essentially the maximal DLs for which conjunctive query answering can be computed in LOGSPACE, and allowing one to delegate query evaluation to a relational engine. Indeed, even slight extensions to *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ make query answering (actually already instance checking, i.e., answering atomic queries) at least NLOGSPACE in data complexity, and thus rule out the possibility of using off-the-shelf relational technology for query processing. In this sense, the *DL-Lite* family includes the first DLs specifically tailored for effective query answering over large amounts of instances.

The results presented in this report are part of our investigation carried out within the entire Work Package 4 ("Ontology access, processing, and usage") on reasoning, and in particular on query answering, over less expressive DLs. Specifically, after establishing how difficult is the problem of query answering (and instance checking) from a computational point of view in the TONES Deliverable D10 [29],

[20] See `http://www.omg.org/uml/`.

we define here techniques for answering of expressive queries (namely, conjunctive queries) over those DLs for which this task is in LOGSPACE, thus allowing for the exploitation of database technique, as mentioned above.

We point out that a first description of some members of the *DL-Lite* family already appeared in some previous TONES deliverables, namely, in the TONES Deliverable D06 "Formalisms for Representing Ontologies: State of the Art Survey" [7], where a first description of *DL-Lite*$_{\mathcal{F}}$ has been provided, and its property have been informally described; in the TONES Deliverable D08 "Common Framework for Representing Ontologies" [33], where a DL called *DL-Lite*$_{\mathcal{A}}$, which generalizes both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, has been presented as an instantiation of the tones framework for stand-alone and situated ontologies; in the TONES Deliverable D13 "Techniques for Ontology Design and Maintenance" [6], where satisfiability, disjointness, and containment of queries posed over *DL-Lite*$_{\mathcal{F}}$ KBs have been discussed. However, in the present report, we provide for the first time a complete picture of *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, the basic DLs of the entire *DL-Lite* family. Also, the technical development on *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ presented in this document constitutes the theoretical foundation for all the logics in this family.

We notice also that the material included in the present document has been recently accepted for publication in an international journal [36].

The rest of this section is organized as follows. The next subsection defines the *DL-Lite* family and the associated reasoning services, arguing that the DLs of this family are indeed interesting logics for capturing the basic modeling constructs of ontology languages. Section 7.2 deals with traditional DL reasoning services for the *DL-Lite* family, such as knowledge base satisfiability, concept and role subsumption, and instance checking. In Section 7.3, we discuss preliminary properties of query answering in the *DL-Lite* family, while in Sections 7.4 and 7.5 we address the problem of query answering for *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, respectively.

7.1 The *DL-Lite* Family

In this section, we focus on a family of DLs, called the *DL-Lite family*. In particular, we deal with three DLs of this family, called *DL-Lite*$_{core}$, *DL-Lite*$_{\mathcal{F}}$, and *DL-Lite*$_{\mathcal{R}}$, respectively.

7.1.1 *DL-Lite*$_{core}$

The language of *DL-Lite*$_{core}$ is the core language for the whole family. Concepts and roles are formed according to the following syntax:

$$\begin{array}{lcl@{\qquad\qquad}lcl} B & \longrightarrow & A \mid \exists R & R & \longrightarrow & P \mid P^- \\ C & \longrightarrow & B \mid \neg B & E & \longrightarrow & R \mid \neg R \end{array}$$

where A denotes an *atomic concept*, P an *atomic role*, and P^- the *inverse* of the atomic role P. B denotes a *basic concept*, i.e., a concept that can be either an atomic concept or a concept of the form $\exists R$, and R denotes a *basic role*, i.e., a role that is either an atomic role or the inverse of an atomic role. Note that $\exists R$ is the standard DL construct of unqualified existential quantification on basic roles. Sometimes we write R^- with the intended meaning that $R^- = P^-$ if $R = P$, and $R^- = P$, if $R = P^-$. Finally, C denotes a *(general) concept*, which can be a basic concept or its negation, whereas E denotes a *(general) role*, which can be a basic role or its negation. Sometimes we write $\neg C$ (resp., $\neg E$) with the intended meaning that $\neg C = \neg A$ if $C = A$ (resp., $\neg E = \neg R$ if $E = R$), and $\neg C = A$, if $C = \neg A$ (resp., $\neg E = R$, if $E = \neg R$).

A DL *knowledge base* (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ represents the domain of interest in terms of two parts, a TBox $\mathcal{T}$, specifying the intensional knowledge, and an ABox $\mathcal{A}$, specifying extensional knowledge.

A *TBox* is formed by a finite set of *inclusion assertions* of the form

$$CL \sqsubseteq C$$

i.e., we allow general concepts to occur on the right-hand side of inclusion assertions, whereas only basic concepts may occur on the left-hand side of inclusion assertions. As we said before, such an inclusion assertion expresses that all instances of concept CL are also instances of concept C. We observe that we might include $CL_1 \sqcup CL_2$ in the constructs for the left-hand side of inclusion assertions (where $\sqcup$ denotes union) and $C_1 \sqcap C_2$ in the constructs for the right-hand side (where $\sqcap$ denotes conjunction). In this way, however, we would not extend the expressive capabilities of the language, since these constructs can be simulated by considering that $CL_1 \sqcup CL_2 \sqsubseteq C$ is equivalent to the pair of assertions $CL_1 \sqsubseteq C$ and $CL_2 \sqsubseteq C$, and that $CL \sqsubseteq C_1 \sqcap C_2$ is equivalent to $CL \sqsubseteq C_1$ and $CL \sqsubseteq C_2$. Similarly, we might add $\bot$ (denoting the empty set) to the constructs for the left-hand side and $\top$ (denoting the whole domain) to those for the right-hand side.

An *ABox* is formed by a finite set of *membership assertions* on atomic concepts and on atomic roles, of the form

$$A(a) \qquad\qquad P(a, b)$$

stating respectively that the object denoted by the constant a is an instance of A and that the pair of objects denoted by the pair of constants (a, b) is an instance of the role P.

In this section, we will not consider membership assertions involving general concepts and roles. However, it can be shown that, with respect to the forms of reasoning considered here, a membership assertion $C(a)$ can be simulated in *DL-Lite*$_{core}$ by adding $A \sqsubseteq C$ to the TBox, and $A(a)$ to the ABox, where A is a new atomic concept. Membership assertions of the form $E(a, b)$, can be simulated with the same mechanisms, in those extensions of *DL-Lite*$_{core}$ (see later) that allow for inclusion assertions on roles.

The semantics of a DL is given in terms of interpretations, where an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$. In particular, for the constructs of *DL-Lite*$_{core}$ we have:

$$\begin{array}{rcl} A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\ P^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (P^-)^{\mathcal{I}} & = & \{(o_2, o_1) \mid (o_1, o_2) \in P^{\mathcal{I}}\} \\ (\exists R)^{\mathcal{I}} & = & \{o \mid \exists o' . (o, o') \in R^{\mathcal{I}}\} \\ (\neg B)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} \\ (\neg R)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}} \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of an inclusion assertion $CL \sqsubseteq C$, if $CL^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. We extend the notion of model also to inclusion assertions of more general forms with respect to the one allowed in *DL-Lite*$_{core}$. An interpretation $\mathcal{I}$ is a *model* of $C_1 \sqsubseteq C_2$, where C_1, C_2 are general concepts, if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. Similarly, $\mathcal{I}$ is a *model* of $E_1 \sqsubseteq E_2$, where E_1, E_2 are general roles, if $E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$.

To specify the semantics of membership assertions, we extend the interpretation function to constants, by assigning to each constant a a *distinct* object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Note that this implies that we enforce the *unique name assumption* on constants [8]. An interpretation $\mathcal{I}$ is a model of a membership assertion $A(a)$, (resp., $P(a, b)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$).

Given an (inclusion, or membership) assertion α, and an interpretation $\mathcal{I}$, we denote by $\mathcal{I} \models \alpha$ the fact that $\mathcal{I}$ is a model of α. Given a (finite) set of assertions κ, we denote by $\mathcal{I} \models \kappa$ the fact that $\mathcal{I}$ is a

model of every assertion in κ. A *model of a KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$. With a little abuse of notation, we also write $\mathcal{I} \models \mathcal{K}$. 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. Similarly, a TBox $\mathcal{T}$ *logically implies* an assertion α, written $\mathcal{T} \models \alpha$, if all models of $\mathcal{T}$ are also models of α.

7.1.2 *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$

We now present two DLs that extend *DL-Lite*$_{core}$, and that are the subject of our investigation in the following sections.

The first DL that we consider is *DL-Lite*$_\mathcal{R}$, which extends *DL-Lite*$_{core}$ with the ability of specifying inclusion assertions between roles of the form

$$R \sqsubseteq E$$

where R and E are defined as above. The semantics of this kind of assertions has been already specified in the previous subsection.

In fact, *DL-Lite*$_\mathcal{R}$ might be enhanced with the capability of managing qualified existential quantification on the right-hand side of inclusion assertions on concepts [35]. This construct, however, can be simulated by suitably making use of inclusions between roles and unqualified existential quantification of concepts in inclusions between concepts, and therefore we do not consider it explicitly.

The second extension of *DL-Lite*$_{core}$ we consider is called *DL-Lite*$_\mathcal{F}$, which extends *DL-Lite*$_{core}$ with the ability of specifying functionality on roles or on their inverses. Assertions used to this aim are of the form

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

where R is defined as above.

An interpretation $\mathcal{I}$ is a model of an assertion $(\mathsf{funct}\ R)$ if the binary relation $R^\mathcal{I}$ is a function, i.e., $(o, o_1) \in R^\mathcal{I}$ and $(o, o_2) \in R^\mathcal{I}$ implies $o_1 = o_2$.

Despite the simplicity of the language and of the assertions allowed, the DLs in the *DL-Lite* family are able to capture the main notions (though not all, obviously) of both ontologies and conceptual modeling formalisms used in databases and software engineering (i.e., Entity-Relationship and UML class diagrams). In particular, *DL-Lite* assertions allow us to specify:

- *ISA*, e.g., stating that concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$;
- *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqsubseteq \neg A_2$;
- *role-typing*, e.g., stating that the first (resp., second) component of the relation P is an instance of A_1 (resp., A_2), using $\exists P \sqsubseteq A_1$ (resp., $\exists P^- \sqsubseteq A_2$);
- *mandatory participation*, e.g., stating that all instances of concept A participate to the relation P as the first (resp., second) component, using $A \sqsubseteq \exists P$ (resp., $A \sqsubseteq \exists P^-$);
- *mandatory non-participation*, using $A \sqsubseteq \neg\exists P$ or $A \sqsubseteq \neg\exists P^-$;
- *functionality restrictions* on roles, using $(\mathsf{funct}\ P)$ or $(\mathsf{funct}\ P^-)$.

Notice that *DL-Lite*$_\mathcal{F}$ is a strict subset of OWL[21] (in fact of OWL Lite). Notice that, the latter includes constructs (e.g., some kinds of role restrictions) and forms of inclusion assertions that are not expressible in *DL-Lite*, and that make reasoning (already in OWL Lite) non-tractable in general.

[21] `http://www.w3.org/TR/owl-features/`

Instead, *DL-Lite*$_{\mathcal{R}}$ can be seen as an extension of (the DL-like part of) the ontology language RDFS[22]. Indeed, *DL-Lite*$_{\mathcal{R}}$ adds to RDFS the ability of expressing disjointness of concepts and roles, and mandatory participation and non-participation.

Example 33. *Consider the atomic concepts* $Professor$ *and* $Student$*, the roles* $TeachesTo$ *and* $HasTutor$*, and the following DL-Lite*$_{core}$ *TBox* $\mathcal{T}$*:*

$$\begin{aligned} Professor &\sqsubseteq \exists TeachesTo \\ Student &\sqsubseteq \exists HasTutor \\ \exists TeachesTo^- &\sqsubseteq Student \\ \exists HasTutor^- &\sqsubseteq Professor \\ Professor &\sqsubseteq \neg Student \end{aligned}$$

Such a TBox states that professors do teach to students, that students have a tutor, which is also a professor, and that no student is also a professor (and vice-versa).

Notice that in DL-Lite$_{\mathcal{R}}$ *we could add the assertion*

$$HasTutor^- \sqsubseteq TeachesTo$$

stating that a tutor also teaches the student s/he is tutoring.

On the other hand, in DL-Lite$_{\mathcal{F}}$ *we could add the assertion*

$$(\mathsf{funct}\ HasTutor)$$

stating that everyone has at most one tutor.

Finally, we show a simple ABox $\mathcal{A}$*:*

$$Student(\mathsf{John}), \quad HasTutor(\mathsf{John}, \mathsf{Mary}), \quad TeachesTo(\mathsf{Mary}, \mathsf{Bill}).$$

■

We conclude this subsection with a brief discussion on the *finite model property* [8]. We remind the reader that a DL enjoys the finite model property if every satisfiable KB expressed in this logic admits a model with a finite domain. It is interesting to observe that, while *DL-Lite*$_{\mathcal{R}}$ has the finite model property [131], *DL-Lite*$_{\mathcal{F}}$ does not, as the following example shows. Consider the *DL-Lite*$_{\mathcal{F}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ with

$$\mathcal{T} = \{\ A \sqsubseteq \exists P, \quad \exists P^- \sqsubseteq A, \quad (\mathsf{funct}\ P^-), \quad B \sqsubseteq \exists P \quad B \sqsubseteq \neg A\ \}$$

and $\mathcal{A} = \{B(a)\}$. It is easy to see that $\mathcal{K}$ admits only infinite models.

7.1.3 Queries

We start with a general notion of queries in first-order logic, and then we move to the definition of queries over a DL KB.

A *query* is an open formula of first-order logic with equalities (FOL in the following). We denote a (FOL) query q as follows

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

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

where $\phi(\vec{x})$ is a FOL formula with free variables $\vec{x}$. We call the size of $\vec{x}$ the *arity* of the query q. Given an interpretation $\mathcal{I}$, $q^{\mathcal{I}}$ is the set of tuples of domain elements that, when assigned to the free variables, make the formula ϕ true in $\mathcal{I}$ [1].

A *boolean query* is a query that does not involve any free variable (i.e., it is a closed formula). Given a boolean query $q = \{ \mid \phi\}$, we may denote it simply by ϕ. Given an interpretation $\mathcal{I}$, $\phi^{\mathcal{I}}$ consists of the only *empty tuple*, i.e., the tuple of arity 0, in the case in which ϕ is true in $\mathcal{I}$, whereas $\phi^{\mathcal{I}}$ is obviously empty if ϕ is false in $\mathcal{I}$.

Among the various queries, we are interested in conjunctive queries and union of conjunctive queries. A *conjunctive query (CQ)* q is a query of the form

$$\{ \vec{x} \mid \exists \vec{y}.conj(\vec{x}, \vec{y}) \}$$

where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms and equalities, with free variables $\vec{x}$ and $\vec{y}$. A *union of conjunctive queries (UCQ)* q is a query of the form

$$\{ \vec{x} \mid \bigvee_{i=1,\ldots,n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i}) \}$$

where each $conj_i(\vec{x}, \vec{y_i})$ is, as before, a conjunction of atoms and equalities with free variables $\vec{x}$ and $\vec{y_i}$. Obviously, the class of union of conjunctive queries contains the class of conjunctive queries.

Sometimes, we use the standard datalog notation (see e.g., [1]) to denote conjunctive queries and unions of conjunctive queries. Namely, a conjunctive query $q = \{ \vec{x} \mid \exists \vec{y}.conj(\vec{x}, \vec{y}) \}$ is denoted in datalog notation as

$$q(\vec{x'}) \, conj'(\vec{x'}, \vec{y'})$$

where $conj'(\vec{x'}, \vec{y'})$ is the list of atoms in $conj(\vec{x}, \vec{y})$ obtained after having equated the variables $\vec{x}, \vec{y}$ according to the equalities in $conj(\vec{x}, \vec{y})$. As a result of such equality elimination, we have that $\vec{x'}$ and $\vec{y'}$ can actually contain constants and multiple occurrences of the same variable. We call $q(\vec{x'})$ the head of q, and $conj'(\vec{x'}, \vec{y'})$ the body. Moreover, we call the variables in $\vec{x'}$ the *distinguished variables* of q and those in $\vec{y'}$ the *non-distinguished variables*.

The datalog notation is then extended to unions of conjunctive queries as follows. A union of conjunctive queries

$$q = \{ \vec{x} \mid \bigvee_{i=1,\ldots,n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i}) \}$$

is denoted in datalog notation as

$$q = \{ q_1, \ldots, q_n \}$$

where each q_i is the datalog expression corresponding to the conjunctive query $q_i = \{ \vec{x} \mid \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i}) \}$.

With the general notion of query in place, we can now define queries over a DL KB. In particular, we will concentrate on conjunctive queries and unions of conjunctive queries. A *conjunctive query over a KB* $\mathcal{K}$ is a conjunctive query whose atoms are of the form $A(z)$ or $P(z_1, z_2)$, where A and P are respectively an atomic concept and an atomic role of $\mathcal{K}$, and z, z_1, z_2 are either constants in $\mathcal{K}$ or variables. Similarly, we define *unions of conjunctive queries over a KB* $\mathcal{K}$.

Given a query q (either a conjunctive query or a union of conjunctive queries) and a KB $\mathcal{K}$, the *answer to* q *over* $\mathcal{K}$ is the set $ans(q, \mathcal{K})$ of tuples $\vec{a}$ of constants appearing in $\mathcal{K}$ such that $\vec{a}^{\mathcal{M}} \in q^{\mathcal{M}}$, for every model $\mathcal{M}$ of $\mathcal{K}$. Notice that by definition $ans(q, \mathcal{K})$ is finite since $\mathcal{K}$ is finite, and hence the number of constants appearing in $\mathcal{K}$ is finite. Notice also that the tuple $\vec{a}$ can be the empty tuple in the case in which q is a boolean conjunctive query. More precisely, in this case the set $ans(q, \mathcal{K})$ consists of the only empty tuple if and only if the formula q is true in every model of $\mathcal{K}$.

Observe that, if $\mathcal{K}$ is unsatisfiable, then $ans(q, \mathcal{K})$ is trivially the set of all possible tuples of constants in $\mathcal{K}$ whose arity is the one of the query. We denote such a set by *AllTup*$(q, \mathcal{K})$.

7.1.4 Reasoning Services

In studying the *DL-Lite* family, we are interested in several reasoning services. Obviously, we want to take into account traditional DL reasoning services, and in particular, for both *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ KBs, we consider the following problems:

- *knowledge base satisfiability*, i.e., given a KB $\mathcal{K}$, verify whether $\mathcal{K}$ admits at least one model;
- *logical implication of KB assertions*, which consists of the following subproblems:
  - *instance checking*, i.e., given a KB $\mathcal{K}$, a concept C and a constant a (resp., a role E and a pair of constants a and b), verify whether $\mathcal{K} \models C(a)$ (resp., $\mathcal{K} \models E(a, b)$);
  - *subsumption of concepts or roles*, i.e., given a TBox $\mathcal{T}$ and two general concepts C_1 and C_2 (resp., two general roles E_1 and E_2), verify whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{T} \models E_1 \sqsubseteq E_2$).
  - *checking functionality*, i.e., given a TBox $\mathcal{T}$ and a basic role R, verify whether $\mathcal{T} \models (\mathsf{funct}\ R)$.

In addition we are interested in:

- *query answering*, i.e., given a KB $\mathcal{K}$ and a query q (either a conjunctive query or a union of conjunctive queries) over $\mathcal{K}$, compute the set $ans(q, \mathcal{K})$.

The following decision problem, called *recognition problem*, is associated to the query answering problem: given a KB $\mathcal{K}$, a query q (either a conjunctive query or a union of conjunctive queries), and a tuple of constants $\vec{a}$ of $\mathcal{K}$, check whether $\vec{a} \in ans(q, \mathcal{K})$. When we talk about the computational complexity of query answering, in fact we implicitly refer to the associated recognition problem.

In analyzing the computational complexity of a reasoning problem over a DL KB, we distinguish between data complexity and combined complexity [148]: *data complexity* is the complexity with respect to the size of the ABox only, while *combined complexity* is the complexity with respect to the size of all inputs to the problem.

7.1.5 The Notion of FOL-Reducibility

We now introduce the notion of FOL-reducibility for both satisfiability and query answering, which will be used in the sequel.

First, given an ABox $\mathcal{A}$ (of the kind considered above), we denote by $db(\mathcal{A}) = \langle \Delta^{db(\mathcal{A})}, \cdot^{db(\mathcal{A})} \rangle$ *the interpretation* defined as follows:

- $\Delta^{db(\mathcal{A})}$ is the non-empty set consisting of all constants occurring in $\mathcal{A}$,
- $a^{db(\mathcal{A})} = a$, for each constant a,
- $A^{db(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept A, and
- $P^{db(\mathcal{A})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, for each atomic role P.

Observe that the interpretation $db(\mathcal{A})$ is a minimal model of the ABox $\mathcal{A}$.

Intuitively, FOL-reducibility of satisfiability (resp., query answering) captures the property that we can reduce satisfiability checking (resp., query answering) to evaluating a FOL query over the ABox $\mathcal{A}$ considered as a relational database, i.e., over $db(\mathcal{A})$. The definitions follow.

Definition 34. Satisfiability in a DL $\mathcal{L}$ is *FOL-reducible*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a boolean FOL query q, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$, $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if q evaluates to false in $db(\mathcal{A})$. △

Definition 35. Query answering in a DL $\mathcal{L}$ for unions of conjunctive queries is *FOL-reducible*, if for every union of conjunctive queries q and every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a FOL query q_1, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$ and every tuple of constants $\vec{a}$ occurring in $\mathcal{A}$, $\vec{a} \in ans(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ if and only if $\vec{a}^{db(\mathcal{A})} \in q_1^{db(\mathcal{A})}$. △

7.2 Techniques for DL Reasoning

In this section, we study traditional DL reasoning services for KBs expressed using the *DL-Lite* family. In particular, we consider knowledge base satisfiability and logical implication, which means (concept/role) instance checking, (concept/role) subsumption, and implication of functionality assertions. We show that all such reasoning services are in PTIME w.r.t. combined complexity, and that instance checking and satisfiability (which make use of the ABox) are FOL-reducible, and hence in LOGSPACE with respect to data complexity. We start our investigation from KB satisfiability, and then we tackle logical implication. In fact, logical implication can be basically reduced to KB satisfiability, and therefore we first give algorithms for KB satisfiability and then we show how to reduce logical implication to such a service. Finally, we provide the complexity results mentioned above.

Hereinafter, we call *positive inclusions (PIs)* assertions of the form $B_1 \sqsubseteq B_2$ or of the form $R_1 \sqsubseteq R_2$, whereas we call *negative inclusions (NIs)* assertions of the form $B_1 \sqsubseteq \neg B_2$ or $R_1 \sqsubseteq \neg R_2$.

7.2.1 Knowledge Base Satisfiability

Our goal in this subsection is to show that knowledge base satisfiability is FOL-reducible. To this aim, we resort to two main constructions, namely the canonical interpretation and the closure of the negative inclusions. We present them in turn below.

Canonical Interpretation The canonical interpretation of a KB expressed either in *DL-Lite*$_\mathcal{R}$ or in *DL-Lite*$_\mathcal{F}$ is an interpretation constructed according to the notion of *chase* [1]. In particular, we adapt here the notion of *restricted chase* adopted by Johnson and Klug in [89].

We start by defining the notion of applicable positive inclusion assertions (PIs), and then we exploit applicable PIs to construct the chase for *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ knowledge bases. Finally, with the notion of chase in place, we give the definition of canonical interpretation.

In the following, as usual, we denote an atomic concept with the symbol A, possibly with subscript, an atomic role with the symbol P, possibly with subscript, and a basic role with the symbol R, possibly with subscript. Furthermore, for easyness of exposition, we make use of the function ga that takes as input a basic role and two constants and returns a membership assertion, as specified below

$$\mathsf{ga}(R, a, b) = \begin{cases} P(a, b), & \text{if } R = P \\ P(b, a), & \text{if } R = P^- \end{cases}$$

Definition 36. Let $\mathcal{S}$ be a set of *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ membership assertions, and let $\mathcal{T}_p$ be a set of *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ PIs. Then, a PI $\alpha \in \mathcal{T}_p$ is *applicable in $\mathcal{S}$ to a membership assertion $f \in \mathcal{S}$* if

- $\alpha = A_1 \sqsubseteq A_2$, $f = A_1(a)$, and $A_2(a) \notin \mathcal{S}$;
- $\alpha = A \sqsubseteq \exists R$, $f = A(a)$, and there does not exist any constant b such that $\mathsf{ga}(R, a, b) \in \mathcal{S}$;
- $\alpha = \exists R \sqsubseteq A$, $f = \mathsf{ga}(R, a, b)$, and $A(a) \notin \mathcal{S}$;
- $\alpha = \exists R_1 \sqsubseteq \exists R_2$, $f = \mathsf{ga}(R_1, a, b)$, and there does not exist any constant c such that $\mathsf{ga}(R_2, a, c) \in \mathcal{S}$;

- $\alpha = R_1 \sqsubseteq R_2$, $f = \mathsf{ga}(R_1, a, b)$, and $\mathsf{ga}(R_2, a, b) \notin \mathcal{S}$.

$\triangle$

Applicable PIs can be used, i.e., *applied*, in order to construct the chase of a KB. Roughly speaking, the chase of a *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a (possibly infinite) set of membership assertions, constructed step-by-step starting from the ABox $\mathcal{A}$. At each step of the construction, a PI $\alpha \in \mathcal{T}$ is applied to a membership assertion f belonging to the current set $\mathcal{S}$ of membership assertions. Applying a PI means adding a new suitable membership assertion to $\mathcal{S}$, thus obtaining a new set $\mathcal{S}'$ in which α is not applicable to f anymore. For example, if $\alpha = A_1 \sqsubseteq A_2$ is applicable in $\mathcal{S}$ to $f = A_1(a)$, the membership assertion to be added to $\mathcal{S}$ is $A_2(a)$, i.e., $\mathcal{S}' = \mathcal{S} \cup A_2(a)$. In some cases (i.e., $\alpha = A \sqsubseteq \exists R$ or $\alpha = \exists R_1 \sqsubseteq \exists R_2$), to achieve an analogous aim, the new membership assertion has to make use of a new constant symbol that does not occur in $\mathcal{S}$.

Notice that such a construction process strongly depends on the order in which we select both the PI to be applied at each step and the membership assertion to which such a PI is applied, as well as on which constants we introduce at each step. Therefore, a number of syntactically distinct sets of membership assertions might result from this process. However, it is possible to show that the result is unique up to renaming of constants occurring in each such a set. Since we want our construction process to come out with a unique chase of a certain knowledge base, along the lines of [89], we assume in the following to have a fixed infinite set of constants, whose symbols are ordered in lexicographic way, and we select PIs, membership assertions and constant symbols in lexicographic order. More precisely, given a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote with Γ_A the set of all constant symbols occurring in $\mathcal{A}$. Also, we assume to have an infinite set Γ_N of constant symbols not occurring in $\mathcal{A}$, such that the set $\Gamma_C = \Gamma_A \cup \Gamma_N$ is totally ordered in lexicographic way. Then, our notion of chase is precisely given below.

Definition 37. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ KB, let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$, let n be the number of membership assertions in $\mathcal{A}$, and let Γ_N be the set of constants defined above. Assume that the membership assertions in $\mathcal{A}$ are numbered from 1 to n following their lexicographic order, and consider the following definition

- $\mathcal{S}_0 = \mathcal{A}$
- $\mathcal{S}_{j+1} = \mathcal{S}_j \cup \{f_{new}\}$, where f_{new} is a membership assertion numbered with $n + j + 1$ in $\mathcal{S}_{j+1}$ and obtained as follows

  let f be the first membership assertion in $\mathcal{S}_j$ such that
  there exists a PI $\alpha \in \mathcal{T}_p$ applicable in $\mathcal{S}_j$ to f
  let α be the lexicographically first PI applicable in $\mathcal{S}_j$ to f
  let a_{new} be the constant of Γ_N that follows lexicographically
  all constants occurring in $\mathcal{S}_j$
  case α, f of
  (cr1) $\alpha = A_1 \sqsubseteq A_2$, $f = A_1(a)$
  then $f_{new} = A_2(a)$
  (cr2) $\alpha = A \sqsubseteq \exists R$ and $f = A(a)$
  then $f_{new} = \mathsf{ga}(R, a, a_{new})$
  (cr3) $\alpha = \exists R \sqsubseteq A$ and $f = \mathsf{ga}(R, a, b)$
  then $f_{new} = A(a)$
  (cr4) $\alpha = \exists R_1 \sqsubseteq \exists R_2$ and $f = \mathsf{ga}(R_1, a, b)$
  then $f_{new} = \mathsf{ga}(R_2, a, a_{new})$
  (cr5) $\alpha = R_1 \sqsubseteq R_2$ and $f = \mathsf{ga}(R_1, a, b)$
  then $f_{new} = \mathsf{ga}(R_2, a, b)$.

Then, we call *chase of* $\mathcal{K}$, denoted $chase(\mathcal{K})$, the set of membership assertions obtained as the infinite union of all $\mathcal{S}_j$, i.e.,

$$chase(\mathcal{K}) = \bigcup_{j \in \mathbb{N}} \mathcal{S}_j.$$

△

In the above definition, **cr1**, **cr2**, **cr3**, **cr4**, and **cr5** indicate the five rules that are used for constructing the chase, each one corresponding to the application of a PI. Such rules are called *chase rules*, and we say that a chase rule is applied to a membership assertion f if the corresponding PI is applied to f. Notice that rules **cr1**, **cr2**, **cr3**, **cr4** are applied in the construction of the chase of both *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ KBs, whereas **cr5** is meaningful only for *DL-Lite*$_\mathcal{R}$ KBs, since PIs of the form $R_1 \sqsubseteq R_2$ do not occur in *DL-Lite*$_\mathcal{F}$ KBs. Observe also that NIs and functionality assertions in $\mathcal{K}$ have no role in constructing $chase(\mathcal{K})$. Indeed $chase(\mathcal{K})$ depends only on the ABox $\mathcal{A}$ and the PIs in $\mathcal{T}$.

In the following, we will denote with $chase_i(\mathcal{K})$ the portion of the chase obtained after i applications of the chase rules, selected according to the ordering established in Definition 37, i.e., $chase_i(\mathcal{K}) = \bigcup_{j \in \{0,..,i\}} \mathcal{S}_j = \mathcal{S}_i$.

The following property shows that the notion of chase of a knowledge base is fair.

Proposition 38. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB, and let α be a PI in $\mathcal{T}$. Then, if there is an $i \in \mathbb{N}$ such that α is applicable in $chase_i(\mathcal{K})$ to a membership assertion $f \in chase_i(\mathcal{K})$, then there is a $j \geq i$ such that $chase_{j+1}(\mathcal{K}) = chase_j(\mathcal{K}) \cup f'$, where f' is the result of applying α to f in $chase_j(\mathcal{K})$.*

With the notion of chase in place we can introduce the notion of canonical interpretation. We define the *canonical interpretation* $can(\mathcal{K})$ as the interpretation $\langle \Delta^{can(\mathcal{K})}, \cdot^{can(\mathcal{K})} \rangle$, where:

- $\Delta^{can(\mathcal{K})} = \Gamma_C$,
- $a^{can(\mathcal{K})} = a$, for each constant a occurring in $chase(\mathcal{K})$,
- $A^{can(\mathcal{K})} = \{a \mid A(a) \in chase(\mathcal{K})\}$, for each atomic concept A, and
- $P^{can(\mathcal{K})} = \{(a_1, a_2) \mid P(a_1, a_2) \in chase(\mathcal{K})\}$, for each atomic role P.

We also define $can_i(\mathcal{K}) = \langle \Delta^{can(\mathcal{K})}, \cdot^{can_i(\mathcal{K})} \rangle$, where $\cdot^{can_i(\mathcal{K})}$ is analogous to $\cdot^{can(\mathcal{K})}$ but it refers to $chase_i(\mathcal{K})$ instead of $chase(\mathcal{K})$. According to the above definition, it is easy to see that $can(\mathcal{K})$ (resp., $can_i(\mathcal{K})$) is unique. Notice also that $can_0(\mathcal{K})$ is tightly related to the interpretation $db(\mathcal{A})$. Indeed, while $\Delta^{db(\mathcal{A})} \subseteq \Delta^{can(\mathcal{K})}$, we have that $\cdot^{db(\mathcal{A})} = \cdot^{can_0(\mathcal{K})}$.

Now, we are ready to show a notable property that holds for $can(\mathcal{K})$.

Lemma 39. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can(\mathcal{K})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$.*

Since $\langle \mathcal{T}_p, \mathcal{A} \rangle$ does not contain NIs, to prove the above Lemma it is sufficient to show that $can(\mathcal{K})$ satisfies all membership assertions in $\mathcal{A}$ and all PIs in $\mathcal{T}_p$. The fact that $can(\mathcal{K})$ satisfies all membership assertions in $\mathcal{A}$ follows from the fact that $\mathcal{A} \subseteq chase(\mathcal{K})$, whereas to prove that $can(\mathcal{K}) \models \mathcal{T}_p$ it is possible proceed by contradiction considering all possible forms of PIs.

As a consequence of Lemma 39, every *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{F}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ with only positive inclusions in the TBox, i.e., such that $\mathcal{T} = \mathcal{T}_p$, is always satisfiable, since we can always construct $can(\mathcal{K})$ which is a model for $\mathcal{K}$. Now, one might ask if and how $can(\mathcal{K})$ can be exploited for checking the satisfiability of a KB with also negative inclusions and, for *DL-Lite*$_\mathcal{F}$ KBs, functionality assertions.

As for functionality assertions, the following lemma shows that, to establish that they are satisfied by $can(\mathcal{K})$, we have to simply verify that the interpretation $db(\mathcal{A})$ satisfies them (and vice-versa).

Lemma 40. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{F}$ KB, and let $\mathcal{T}_f$ be the set of functionality assertions in $\mathcal{T}$. Then, $can(\mathcal{K})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ if and only if $db(\mathcal{A})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$.*

The "only if direction" of the above Lemma is easy to prove, whereas for the "if direction" it is possible to proceed by induction on the construction of $chase(\mathcal{K})$.

NI-closure Let us now consider negative inclusions. In particular, we look for a property which is analogous to Lemma 40 for the case of NIs. Notice that, in this case, even if $db(\mathcal{A})$ satisfies the NIs asserted in the KB $\mathcal{K} = \langle \mathcal{T}, A \rangle$, we have that $can(\mathcal{K})$ may not satisfy $\mathcal{K}$. For example, if $\mathcal{T}$ contains the inclusion assertions $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq \neg A_3$, and $\mathcal{A}$ contains the membership assertions $A_1(a)$ and $A_3(a)$, it is easy to see that $db(\mathcal{A}) \models A_2 \sqsubseteq \neg A_3$, but $can(\mathcal{K}) \not\models A_2 \sqsubseteq \neg A_3$.

However, as suggested by the simple example above, we get that to find the property we are looking for, we need to properly take into account the interaction between positive and negative inclusions. To this aim we construct a special TBox by closing the NIs with respect to the PIs.

Definition 41. Let $\mathcal{T}$ be a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ TBox. We call *NI-closure of* $\mathcal{T}$, denoted by $cln(\mathcal{T})$, the TBox defined inductively as follows:

1. all negative inclusion assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
2. all functionality assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;
3. if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ is in $cln(\mathcal{T})$, then also $B_1 \sqsubseteq \neg B_3$ is in $cln(\mathcal{T})$;
4. if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $\exists R_2 \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists R_2$ is in $cln(\mathcal{T})$, then also $\exists R_1 \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;
5. if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $\exists R_2^- \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists R_2^-$ is in $cln(\mathcal{T})$, then also $\exists R_1^- \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;
6. if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $R_2 \sqsubseteq \neg R_3$ or $R_3 \sqsubseteq \neg R_2$ is in $cln(\mathcal{T})$, then also $R_1 \sqsubseteq \neg R_3$ is in $cln(\mathcal{T})$.
7. (a) in the case in which $\mathcal{T}$ is a *DL-Lite*$_\mathcal{F}$ TBox, if one of the assertions $\exists R \sqsubseteq \neg \exists R$, or $\exists R^- \sqsubseteq \neg \exists R^-$ is in $cln(\mathcal{T})$, then both such assertions are in $cln(\mathcal{T})$;
   (b) in the case in which $\mathcal{T}$ s a *DL-Lite*$_\mathcal{R}$ TBox, if one of the assertions $\exists R \sqsubseteq \neg \exists R$, $\exists R^- \sqsubseteq \neg \exists R^-$, or $R \sqsubseteq \neg R$ is in $cln(\mathcal{T})$, then all three such assertions are in $cln(\mathcal{T})$.

△

Notice that in the construction of the NI-closure of *DL-Lite*$_\mathcal{F}$ TBoxes, we make use only of Rules 1, 2, 3, and 7(a) whereas for *DL-Lite*$_\mathcal{R}$ TBoxes, we make use only of Rules 1, 3, 4, 5, 6, and 7(b).

It is possible then to prove that, provided we have computed $cln(\mathcal{T})$, the analogous of Lemma 39 and Lemma 40 holds also for NIs.

Lemma 42. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB. Then, $can(\mathcal{K})$ is a model of $\mathcal{K}$ if and only if $db(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

As for proving the "only if direction" of the above Lemma, we exploit the property that $cln(\mathcal{T})$ for *DL-Lite*$_\mathcal{R}$ (resp. *DL-Lite*$_\mathcal{F}$) KBs does not imply new negative inclusions (resp. negative inclusions and functionality assertions) not implied by $\mathcal{T}$. The "if direction" can be proved by induction on the construction of the chase.

The following corollary is an interesting consequence of the lemma above.

Corollary 43. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ TBox, and α a negative inclusion assertion or a functionality assertion. We have that, if $\mathcal{T} \models \alpha$, then $cln(\mathcal{T}) \models \alpha$.*

FOL-Reducibility Before providing the main theorems of this subsection, we need also the following property, which asserts that to establish satisfiability of a knowledge base, we can resort to constructing the canonical interpretation.

Lemma 44. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB. Then, $can(\mathcal{K})$ is a model of $\mathcal{K}$ if and only if $\mathcal{K}$ is satisfiable.*

The "only if direction" is of the above lemma is trivial, whereas for the "if direction" we exploit Lemma 42 and the property that $cln(\mathcal{T})$ for *DL-Lite*$_\mathcal{R}$ (resp. *DL-Lite*$_\mathcal{F}$) KBs does not imply new negative inclusions (resp. negative inclusions and functionality assertions) not implied by $\mathcal{T}$.

Notice that, the construction of $can(\mathcal{K})$ is in general neither convenient nor possible, since $can(\mathcal{K})$ may be infinite. However, by simply combining Lemma 42 and Lemma 44, we obtain the notable result that to check satisfiability of a knowledge base, it is sufficient (and necessary) to look at $db(\mathcal{A})$ (provided we have computed $cln(\mathcal{T})$). More precisely, the next theorem shows that a contradiction on a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ KB may hold only if a membership assertion in the ABox contradicts a functionality assertion or a NI implied by the closure $cln(\mathcal{T})$.

Theorem 45. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB. Then, $\mathcal{K}$ is satisfiable if and only if $db(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

At this point, it is not difficult to show that verifying if $db(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$ can be done by simply evaluating a suitable boolean FOL query over $db(\mathcal{A})$ itself. In particular we define a translation function δ from assertions in $cln(\mathcal{T})$ to FOL formulas, as follows:

$$\begin{aligned}
\delta((\mathsf{funct}\ P)) &= \exists x, y, z.P(x, y) \wedge P(x, z) \wedge y \neq z \\
\delta((\mathsf{funct}\ P^-)) &= \exists x, y, z.P(x, y) \wedge P(z, y) \wedge x \neq z \\
\delta(B_1 \sqsubseteq \neg B_2) &= \exists x.\gamma_1(x) \wedge \gamma_2(x) \\
\delta(R_1 \sqsubseteq \neg R_2) &= \exists x, y.\rho_1(x, y) \wedge \rho_2(x, y)
\end{aligned}$$

where in the last equations $\gamma_i(x) = A_i(x)$ if $B_i = A_i$, $\gamma_i(x) = \exists y_i.P_i(x, y_i)$ if $B_i = \exists P_i$, and $\gamma_i(x) = \exists y_i.P_i(y_i, x)$ if $B_i = \exists P_i^-$; and $\rho_i(x, y) = P_i(x, y)$ if $R_i = P_i$, and $\rho_i(x, y) = P_i(y, x)$ if $R_i = P_i^-$.

The algorithm Consistent, described in Figure 9, takes as input a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ KB, computes $db(\mathcal{A})$ and $cln(\mathcal{T})$, and evaluates over $db(\mathcal{A})$ the boolean FOL query obtained by taking the union of all FOL formulas returned by the application of the above function δ to every assertion in $cln(\mathcal{T})$[23]. In the algorithm, the symbol $\bot$ indicates a predicate whose evaluation is $false$ in every interpretation. Therefore, in the case in which neither functionality assertions nor negative inclusion assertions occur in $\mathcal{K}$, $q_{unsat}^{db(\mathcal{A})} = \bot^{db(\mathcal{A})}$, and therefore Consistent($\mathcal{K}$) returns $true$.

Lemma 46. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB. Then, the algorithm Consistent($\mathcal{K}$) terminates, and $\mathcal{K}$ is satisfiable if and only if Consistent($\mathcal{K}$) $= true$.*

Proof. Since $cln(\mathcal{T})$ is a finite set of membership and functionality assertions, the algorithm terminates. By Theorem 45, we have that $db(\mathcal{A})$ is a model of all assertions in $cln(\mathcal{T})$ if and only if $\mathcal{K}$ is satisfiable. The query q_{unsat} verifies whether there exists an assertion α that is violated in $db(\mathcal{A})$, by expressing its negation as a FOL formula $\delta(\alpha)$ and evaluating it in $db(\mathcal{A})$. ❏

As a direct consequence of Lemma 46, we get:

Theorem 47. *Knowledge base satisfiability in DL-Lite$_\mathcal{R}$ and DL-Lite$_\mathcal{F}$ is FOL-reducible.*

[23] A first version of the algorithm Consistent, limited to checking satisfiability of only *DL-Lite*$_\mathcal{F}$ KBs, has been already presented in [6].

Algorithm Consistent($\mathcal{K}$)
Input: *DL-Lite* knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: $true$ if $\mathcal{K}$ is satisfiable, $false$ otherwise
begin
  $q_{unsat} = \bot$;
  for each $\alpha \in cln(\mathcal{T})$ **do**
    $q_{unsat} = q_{unsat} \vee \delta(\alpha)$;
  if $q_{unsat}^{db(\mathcal{A})} = \emptyset$ **return** $true$;
  else return $false$;
end

Figure 9: The algorithm Consistent

Example 48. *We now check satisfiability of the DL-Lite$_{core}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ in Example 33. To this aim, we first compute $cln(\mathcal{T})$, which is as follows:*

$$\begin{aligned} Professor &\sqsubseteq \neg Student \\ \exists TeachesTo^- &\sqsubseteq \neg Professor \\ \exists HasTutor^- &\sqsubseteq \neg Student. \end{aligned}$$

Next, we apply the translation function δ to each NI above, getting:

$$\begin{aligned} &\delta(Professor \sqsubseteq \neg Student) = \exists x.Professor(x) \wedge Student(x) \\ &\delta(\exists TeachesTo^- \sqsubseteq \neg Professor) = \exists x.(\exists y.TeachesTo(y,x)) \wedge Professor(x) \\ &\delta(\exists HasTutor^- \sqsubseteq \neg Student) = \exists x.(\exists y.HasTutor(y,x)) \wedge Student(x). \end{aligned}$$

The union of such queries is q_{unsat}, which evaluated over $db(\mathcal{A})$ returns false, thus showing satisfiability of $\mathcal{K}$.

As a further example, consider now the DL-Lite$_{\mathcal{R}}$ TBox $\mathcal{T}'$ obtained from $\mathcal{T}$ by adding the inclusion assertion $HasTutor^- \sqsubseteq TeachesTo$. In this case $cln(\mathcal{T}')$ includes $cln(\mathcal{T})$ plus the following NIs:

$$\begin{aligned} \exists HasTutor &\sqsubseteq \neg Professor \\ \exists TeachesTo &\sqsubseteq \neg Student. \end{aligned}$$

So q'_{unsat} includes the disjuncts of q_{unsat} plus the following:

$$\begin{aligned} &\delta(\exists HasTutor \sqsubseteq \neg Professor) = \exists x.(\exists y.HasTutor(x,y)) \wedge Professor(x) \\ &\delta(\exists TeachesTo \sqsubseteq \neg Student) = \exists x.(\exists y.TeachesTo(x,y)) \wedge Student(x). \end{aligned}$$

Since $(q'_{unsat})^{db(\mathcal{A})}$ is false, we conclude that $\mathcal{K}' = \langle \mathcal{T}', \mathcal{A} \rangle$ is satisfiable.

Finally, if we instead add the functionality assertion (funct $HasTutor$) *to $\mathcal{T}$, we obtain a DL-Lite$_{\mathcal{F}}$ TBox $\mathcal{T}''$, whose NI-closure $cln(\mathcal{T}'')$ includes $cln(\mathcal{T})$ plus* (funct $HasTutor$).

In this case, q''_{unsat} includes the disjuncts of q_{unsat} plus:

$$\delta((\text{funct } HasTutor)) = \exists x, y, z.HasTutor(x,y) \wedge HasTutor(x,z) \wedge y \neq z.$$

Again, $(q''_{unsat})^{db(\mathcal{A})}$ is false, and hence also $\mathcal{K}'' = \langle \mathcal{T}'', \mathcal{A} \rangle$ is satisfiable. ∎

7.2.2 Logical Implication

We start by showing that both instance checking and subsumption can be reduced to knowledge base satisfiability. We first consider the problem of instance checking for concept expressions, and provide a suitable reduction from this problem to knowledge base satisfiability.

Theorem 49. *Let $\mathcal{K}$ be either a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB, C a general concept, d a constant appearing in $\mathcal{K}$, and $\overline{A}$ an atomic concept not appearing in $\mathcal{K}$. Then $\mathcal{K} \models C(d)$ if and only if the KB*

$$\mathcal{K}_{C(d)} = \langle \mathcal{K} \cup \{\overline{A} \sqsubseteq \neg C\},\ \{\overline{A}(d)\}\rangle$$

is unsatisfiable.

The analogous of the above theorem holds for the problem of instance checking for role expressions. We first consider *DL-Lite$_\mathcal{R}$* KBs.

Theorem 50. *Let $\mathcal{K}$ be a DL-Lite$_\mathcal{R}$ KB, E a general role, a and b two constants appearing in $\mathcal{K}$, and $\overline{P}$ an atomic role not appearing in $\mathcal{K}$. Then $\mathcal{K} \models E(a, b)$ if and only if the KB*

$$\mathcal{K}_{E(a,b)} = \langle \mathcal{K} \cup \{\overline{P} \sqsubseteq \neg E\},\ \{\overline{P}(a, b)\}\rangle$$

is unsatisfiable.

Also, for *DL-Lite$_\mathcal{F}$* KBs we provide the following theorem.

Theorem 51. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_\mathcal{F}$ KB, R a basic role, and a and b two constants appearing in $\mathcal{K}$. Then*

- *$\mathcal{K} \models R(a, b)$ if and only if either $\mathcal{K}$ is unsatisfiable, or $\mathsf{ga}(R, a, b) \in \mathcal{A}$.*
- *$\mathcal{K} \models \neg R(a, b)$ if and only if $\langle \mathcal{T}, \mathcal{A} \cup \{\mathsf{ga}(R, a, b)\}\rangle$ is unsatisfiable.*

In the above theorem, the second item and the "if" direction of the first item are obvious. As for the "only-if" direction of the first item, it can be easily reached the contradiction that both $can(\mathcal{K}) \not\models R(a, b)$ and $can(\mathcal{K}) \models R(a, b)$ starting from the (wrong) assumption that $\mathcal{K} \models R(a, b)$ but $\mathcal{K}$ is satisfiable and $\mathsf{ga}(R, a, b) \notin \mathcal{A}$.

We now address the subsumption problem and provide different reductions of this problem to the problem of knowledge base satisfiability. The case of subsumption between concepts is dealt with by the following theorem, and the case of subsumption between roles, is considered in the two subsequent theorems.

Theorem 52. *Let $\mathcal{T}$ be either a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ TBox, C_1 and C_2 two general concepts, $\overline{A}$ an atomic concept not appearing in $\mathcal{T}$, and d a constant. Then, $\mathcal{T} \models C_1 \sqsubseteq C_2$ if and only if the KB*

$$\mathcal{K}_{C_1 \sqsubseteq C_2} = \langle \mathcal{T} \cup \{\overline{A} \sqsubseteq C_1, \overline{A} \sqsubseteq \neg C_2\},\ \{\overline{A}(d)\}\rangle,$$

is unsatisfiable.

Theorem 53. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{R}$ TBox, E_1 and E_2 two general roles, $\overline{P}$ an atomic role not appearing in $\mathcal{T}$, and a, b two constants. Then, $\mathcal{T} \models E_1 \sqsubseteq E_2$ if and only if the KB*

$$\mathcal{K}_{E_1 \sqsubseteq E_2} = \langle \mathcal{T} \cup \{\overline{P} \sqsubseteq E_1, \overline{P} \sqsubseteq \neg E_2\},\ \{\overline{P}(a, b)\}\rangle$$

is unsatisfiable.

Theorem 54. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{F}$ TBox, R_1 and R_2 two basic roles, $\overline{A}$ an atomic concept not appearing in $\mathcal{T}$, and d a constant. Then,*

1. *$\mathcal{T} \models R_1 \sqsubseteq R_2$ or $\mathcal{T} \models \neg R_1 \sqsubseteq \neg R_2$ if and only if (a) $R_1 = R_2$, or (b) the KB*

$$\mathcal{K}_{\exists R_1 \sqsubseteq \neg \exists R_1} = \langle \mathcal{T} \cup \{\overline{A} \sqsubseteq \exists R_1\},\ \{\overline{A}(d)\}\rangle$$

   is unsatisfiable, or (c) the KB

$$\mathcal{K}_{\exists R_1^- \sqsubseteq \neg \exists R_1^-} = \langle \mathcal{T} \cup \{\overline{A} \sqsubseteq \exists R_1^-\},\ \{\overline{A}(d)\}\rangle$$

   is unsatisfiable;

2. *$\mathcal{T} \models \neg R_1 \sqsubseteq R_2$ if and only if $\mathcal{T}$ is unsatisfiable.*

3. *$\mathcal{T} \models R_1 \sqsubseteq \neg R_2$ if and only if either (a) the KB*

$$\mathcal{K}_{\exists R_1 \sqsubseteq \neg \exists R_2} = \langle \mathcal{T} \cup \{\overline{A} \sqsubseteq \exists R_1, \overline{A} \sqsubseteq \exists R_2\},\ \{\overline{A}(d)\}\rangle$$

   is unsatisfiable, or (b) the KB

$$\mathcal{K}^-_{\exists R_1 \sqsubseteq \neg \exists R_2} = \langle \mathcal{T} \cup \{\overline{A} \sqsubseteq \exists R_1^-, \overline{A} \sqsubseteq \exists R_2^-\},\ \{\overline{A}(d)\}\rangle,$$

   is unsatisfiable.

Notice that, according to item 1 of the above theorem, a *DL-Lite$_\mathcal{F}$* TBox $\mathcal{T}$ implies a subsumption $R_1 \sqsubseteq R_2$ if and only if $R_1 = R_2$ or either the subsumption $\exists R_1 \sqsubseteq \neg \exists R_1$ or the subsumption $\exists R_1^- \sqsubseteq \neg \exists R_1^-$ is implied by $\mathcal{T}$. Similarly for item 3, whereas, according to item 2, a *DL-Lite$_\mathcal{F}$* TBOX $\mathcal{T}$ implies a subsumption of the form $\neg R_1 \sqsubseteq R_2$ if and only if $\mathcal{T}$ is unsatisfiable.

The following theorem characterizes logical implication of a functionality assertion in *DL-Lite$_\mathcal{R}$* and *DL-Lite$_\mathcal{F}$*, in terms of subsumption between roles.

Theorem 55. *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ TBox and R a basic role. Then, $\mathcal{T} \models (\mathsf{funct}\ R)$ if and only if either $(\mathsf{funct}\ R) \in \mathcal{T}$ (in the case where $\mathcal{T}$ is a DL-Lite$_\mathcal{F}$ KB), or $\mathcal{T} \models R \sqsubseteq \neg R$.*

Notice that the role inclusion assertion we are using in Theorem 55 is of the form $\mathcal{T} \models R \sqsubseteq \neg R$, and thus expresses the fact that role R has an empty extension in every model of $\mathcal{T}$. Also, by Theorems 53 and 54, logical implication of role inclusion assertions can in turn be reduced to knowledge base satisfiability.

7.2.3 Computational Complexity

From the results in the previous subsections we can establish the computational complexity characterization for the classical DL reasoning problems for *DL-Lite$_\mathcal{R}$* and *DL-Lite$_\mathcal{F}$*.

Theorem 56. *In DL-Lite$_\mathcal{R}$ and DL-Lite$_\mathcal{F}$, knowledge base satisfiability is* LOGSPACE *in the size of the ABox (data complexity) and* PTIME *in the size of the whole knowledge base (combined complexity).*

Proof. First, LOGSPACE data complexity follows directly from FOL-reducibility, since evaluating FOL queries/formulas over a model is LOGSPACE in the size of the model [148]. As for the combined complexity, we have that $cln(\mathcal{T})$ is polynomially related to the size of the TBox $\mathcal{T}$ and hence q_{unsat} defined above is formed by a number of disjuncts that is polynomial in $\mathcal{T}$. Each disjunct can be evaluated separately and contains either 2 or 3 variables. Now, each disjunct can be evaluated by checking the formula under each of the n^3 possible assignments, where n is the size of the domain of $db(\mathcal{A})$ [148]. Finally, once an assignment is fixed the evaluation of the formula can be done in LOGSPACE [148]. As a result, we get the PTIME bound. ❑

Taking into account the reductions in Theorems 49, 50, 51, 52, 53, 54, and 55, as a consequence of Lemma 46, we get the following results.

Theorem 57. *In DL-Lite$_\mathcal{R}$ and DL-Lite$_\mathcal{F}$, (concept/role) subsumption and logical implication of functionality assertions are both* PTIME *in the size of the TBox, while (concept/role) instance checking is* LOGSPACE *in the size of the ABox and* PTIME *in the size of the whole knowledge base.*

7.3 Query Answering: Preliminary Properties

In this subsection we start studying query answering in *DL-Lite$_\mathcal{R}$* and *DL-Lite$_\mathcal{F}$*, and establish some preliminary properties which will be used in the next subsection.

First, we recall that, in the case where $\mathcal{K}$ is an unsatisfiable KB, the answer to a union of conjunctive queries Q is defined as the finite set of tuples *AllTup*$(Q, \mathcal{K})$. Therefore, in the following we focus on the case where $\mathcal{K}$ is satisfiable.

Then, let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable *DL-Lite$_\mathcal{R}$* or *DL-Lite$_\mathcal{F}$* KB, it is possible to show, by induction on the construction of $chase(\mathcal{K})$, that, for every model $\mathcal{M}$ of $\mathcal{K}$, there is a homomorphism from $can(\mathcal{K})$ to $\mathcal{M}$ that maps the objects in the extension of concepts and roles in $can(\mathcal{K})$ to objects in the extension of concepts and roles in $\mathcal{M}$.

Based on the above property, we now prove that the canonical model $can(\mathcal{K})$ of a satisfiable KB $\mathcal{K}$ is able to represent all models of $\mathcal{K}$ with respect to unions of conjunctive queries.

Theorem 58. *Let $\mathcal{K}$ be a satisfiable DL-Lite$_\mathcal{F}$ or DL-Lite$_\mathcal{R}$ KB, and let Q be a union of conjunctive queries over $\mathcal{K}$. Then, $ans(Q, \mathcal{K}) = Q^{can(\mathcal{K})}$.*

The above property shows that the canonical model $can(\mathcal{K})$ is a correct representative of all the models of a *DL-Lite$_\mathcal{R}$* (or *DL-Lite$_\mathcal{F}$*) KB with respect to the problem of answering unions of conjunctive queries. In other words, for every union of conjunctive queries Q, the answers to Q over $\mathcal{K}$ correspond to the evaluation of Q in $can(\mathcal{K})$.

In fact, this property holds for all positive FOL queries, but not in general. Consider for example the *DL-Lite$_{core}$* KB $\mathcal{K} = \langle \emptyset, \{\mathcal{A}_1(d)\} \rangle$, and the FOL boolean query $q = \{ \mid \exists x.A_1(x) \wedge \neg A_2(x)\}$. We have that $chase(\mathcal{K}) = \{\mathcal{A}_1(d)\}$, and therefore q is true in $can(\mathcal{K})$, but the answer to q over $\mathcal{K}$ is false, since there exists a model $\mathcal{M}$ for $\mathcal{K}$ such that q is false in $\mathcal{M}$. Assume, for instance, that $\mathcal{M}$ has the same interpretation domain as $can(\mathcal{K})$, and that $a^\mathcal{M} = a$, $A_1^\mathcal{M} = \{a\}$, and $A_2^\mathcal{M} = \{a\}$. It is easy to see that $\mathcal{M}$ is a model for $\mathcal{K}$ and q is false in $\mathcal{M}$.

We point out that the canonical interpretation is in general infinite, consequently it cannot be effectively computed in order to solve the query answering problem in *DL-Lite$_\mathcal{F}$* or *DL-Lite$_\mathcal{R}$*.

Now, given the limited expressive power of *DL-Lite$_\mathcal{R}$* and *DL-Lite$_\mathcal{F}$* TBoxes, it might seem that, in order to answer a query over a KB $\mathcal{K}$, we could simply build a *finite* interpretation $\mathcal{I}_\mathcal{K}$ that allows for reducing answering *every* union of conjunctive queries (or even every single conjunctive query) over $\mathcal{K}$ to evaluating the query in $\mathcal{I}_\mathcal{K}$. The following theorem shows that this is *not* the case.

Theorem 59. *There exists a DL-Lite$_{core}$ KB $\mathcal{K}$ for which no finite interpretation $\mathcal{I}_\mathcal{K}$ exists such that, for every conjunctive query q over $\mathcal{K}$, $ans(q, \mathcal{K}) = q^{\mathcal{I}_\mathcal{K}}$.*

Proof. Let $\mathcal{K}$ be the *DL-Lite$_{core}$* KB whose TBox consists of the cyclic concept inclusion $\exists P^- \sqsubseteq \exists P$ and whose ABox consists of the assertion $P(a, b)$.

Let $\mathcal{I}_\mathcal{K}$ be a finite interpretation. There are two possible cases:

1. There is no cycle on the relation P in $\mathcal{I}_\mathcal{K}$, i.e., the maximum path on the relation $P^{\mathcal{I}_\mathcal{K}}$ has a finite length n. In this case, consider the boolean conjunctive query $qP(x_1, x_2), \ldots, P(x_n, x_{n+1})$ that

represents the existence of a path of length $n+1$ in P. It is immediate to verify that the query q is false in $\mathcal{I}_\mathcal{K}$, i.e., $q^{\mathcal{I}_\mathcal{K}} = \emptyset$, while the answer to q over $\mathcal{K}$ is true, i.e., $ans(q, \mathcal{K}) \neq \emptyset$ (indeed $ans(q, \mathcal{K})$ consists of the empty tuple). This last property can be easily seen by noticing that $q^{can(\mathcal{K})}$ is true.

2. $\mathcal{I}_\mathcal{K}$ satisfies the TBox cycle, so it has a finite cycle. More precisely, let us assume that $\mathcal{I}_\mathcal{K}$ is such that $(o_1, o_2) \in P^{\mathcal{I}_\mathcal{K}}, (o_2, o_3) \in P^{\mathcal{I}_\mathcal{K}}, \ldots, (o_n, o_1) \in P^{\mathcal{I}_\mathcal{K}}$. In this case, consider the boolean conjunctive query $qP(x_1, x_2), \ldots, P(x_n, x_1)$. It is immediate to verify that such a query is true in $\mathcal{I}_\mathcal{K}$, while the answer to q over $\mathcal{K}$ is false. This last property can be easily seen by noticing that $q^{can(\mathcal{K})}$ is false, since $chase(\mathcal{K})$ does not contain a set of facts $P(a_1, a_2), P(a_2, a_3), \ldots, P(a_n, a_1)$, for any n, and therefore in $can(\mathcal{K})$ there does not exist any cycle on the relation P.

Consequently, in both cases $ans(q, \mathcal{K}) \neq q^{\mathcal{I}_\mathcal{K}}$. ❑

The above property demonstrates that answering queries in *DL-Lite*$_{core}$, and hence both in *DL-Lite*$_\mathcal{R}$ and in *DL-Lite*$_\mathcal{F}$, goes beyond both propositional logic and relational databases.

Finally, we prove a property that relates answering unions of conjunctive queries to answering conjunctive queries.

Theorem 60. *Let $\mathcal{K}$ be either a DL-Lite$_\mathcal{R}$ or a DL-Lite$_\mathcal{F}$ KB, and let Q be a union of conjunctive queries over $\mathcal{K}$. Then, $ans(Q, \mathcal{K}) = \bigcup_{q_i \in Q} ans(q_i, \mathcal{K})$.*

Informally, the above property states that the set of answers to a union of conjunctive queries Q in *DL-Lite*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ corresponds to the union of the answers to the various conjunctive queries in Q.

7.4 Query Answering in *DL-Lite*$_\mathcal{R}$

In this subsection we discuss query answering in *DL-Lite*$_\mathcal{R}$. More precisely, based on the properties shown in the previous subsection, we define an algorithm for answering unions of conjunctive queries in *DL-Lite*$_\mathcal{R}$, and analyze its computational complexity.

In a nutshell, our query answering method strongly separates the intensional and the extensional level of the *DL-Lite*$_\mathcal{R}$ KB: the query is first processed and reformulated based on the TBox axioms; then, the TBox is discarded and the reformulated query is evaluated over the ABox, as if the ABox were a simple relational database (cf. subsection 7.1.5). More precisely, given a query q over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compile the assertions of $\mathcal{T}$ (in fact, the PIs in $\mathcal{T}$) into the query itself, thus obtaining a new query q'. Such a new query q' is then evaluated over $db(\mathcal{A})$, thus essentially reducing query answering to query evaluation over a database instance. Since the size of q' does not depend on the ABox, the data complexity of the whole query answering algorithm is the same as the data complexity of evaluating q'. We show that, in the case where q is a conjunctive query, the query q' is a union of conjunctive queries. Hence, the data complexity of the whole query answering algorithm is polynomial.

In the following, we first define an algorithm for the reformulation of conjunctive queries. Then, we describe a technique for answering unions of conjunctive queries in *DL-Lite*$_\mathcal{R}$ and we prove correctness of such a technique. Finally, we analyze the computational complexity of query answering over *DL-Lite*$_\mathcal{R}$ KBs.

7.4.1 Query Reformulation

We start our presentation by giving some preliminary definitions.

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant.

Instead, an argument of an atom in a query is *unbound* if it corresponds to a non-distinguished non-shared variable. As usual, we use the symbol '_' to represent non-distinguished non-shared variables.

A PI I is *applicable to an atom* $A(x)$, if I has A in its right-hand side.

A PI I is *applicable to an atom* $P(x_1, x_2)$, if: (*i*) $x_2 = \_$ and the right-hand side of I is $\exists P$; or (*ii*) $x_1 = \_$ and the right-hand side of I is $\exists P^-$; or (*iii*) I is a role inclusion assertion and its right-hand side is either P or P^-. Roughly speaking, an inclusion I is applicable to an atom g if the predicate of g is equal to the predicate in the right-hand side of I and, in the case when I is an inclusion assertion between concepts, if g has at most one bound argument and corresponds to the object that is implicitly referred to by the inclusion I.

We indicate with $gr(g, I)$ the atom obtained from the atom g by applying the applicable inclusion I. Formally:

Definition 61. Let I be an inclusion assertion that is applicable to the atom g. Then, $gr(g, I)$ is the atom defined as follows:

- if $g = A(x)$ and $I = A_1 \sqsubseteq A$, then $gr(g, I) = A_1(x)$;
- if $g = A(x)$ and $I = \exists P \sqsubseteq A$, then $gr(g, I) = P(x, \_)$;
- if $g = A(x)$ and $I = \exists P^- \sqsubseteq A$, then $gr(g, I) = P(\_, x)$;
- if $g = P(x, \_)$ and $I = A \sqsubseteq \exists P$, then $gr(g, I) = A(x)$;
- if $g = P(x, \_)$ and $I = \exists P_1 \sqsubseteq \exists P$, then $gr(g, I) = P_1(x, \_)$;
- if $g = P(x, \_)$ and $I = \exists P_1^- \sqsubseteq \exists P$, then $gr(g, I) = P_1(\_, x)$;
- if $g = P(\_, x)$ and $I = A \sqsubseteq \exists P^-$, then $gr(g, I) = A(x)$;
- if $g = P(\_, x)$ and $I = \exists P_1 \sqsubseteq \exists P^-$, then $gr(g, I) = P_1(x, \_)$;
- if $g = P(\_, x)$ and $I = \exists P_1^- \sqsubseteq \exists P^-$, then $gr(g, I) = P_1(\_, x)$;
- if $g = P(x_1, x_2)$ and either $I = P_1 \sqsubseteq P$ or $I = P_1^- \sqsubseteq P^-$, then $gr(g, I) = P_1(x_1, x_2)$;
- if $g = P(x_1, x_2)$ and either $I = P_1 \sqsubseteq P^-$ or $P_1^- \sqsubseteq P$, then $gr(g, I) = P_1(x_2, x_1)$.

△

In Figure 10, we provide the algorithm PerfectRef, which reformulates a conjunctive query taking into account the PIs of a TBox $\mathcal{T}$.

In the algorithm, $q[g/g']$ denotes the conjunctive query obtained from q by replacing the atom g with a new atom g'. Furthermore, τ is a function that takes as input a conjunctive query q and returns a new conjunctive query obtained by replacing each occurrence of an unbound variable in q with the symbol _. Finally, *reduce* is a function that takes as input a conjunctive query q and two atoms g_1 and g_2 occurring in the body of q, and returns a conjunctive query q' obtained by applying to q the *most general unifier* between g_1 and g_2. We point out that, in unifying g_1 and g_2, each occurrence of the _ symbol has to be considered a different unbound variable. The most general unifier substitutes each _ symbol in g_1 with the corresponding argument in g_2, and vice-versa (obviously, if both arguments are _, the resulting argument is _).

Informally, the algorithm first reformulates the atoms of each conjunctive query $q \in PR'$, and produces a new query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, which allow one to compile away in the reformulation the intensional knowledge (represented by $\mathcal{T}$) that is relevant for answering q. At step (b), for each pair of atoms g_1, g_2 that unify and occur in the body of a query q, the algorithm computes the conjunctive query $q' = reduce(q, g_1, g_2)$. Thanks to the unification performed by *reduce*, variables that are bound in q may become unbound in q'. Hence, PIs that were not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (a)). Notice that the use of τ is necessary in order to guarantee that each unbound variable is represented by the symbol _.

Algorithm PerfectRef $(q, \mathcal{T})$
Input: conjunctive query q, TBox $\mathcal{T}$
Output: union of conjunctive queries PR
$PR := \{q\}$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ **do**
  (a) **for each** g in q **do**
      for each PI I in $\mathcal{T}$ **do**
        if I is applicable to g
        then $PR := PR \cup \{\, q[g/gr(g, I)] \,\}$
  (b) **for each** g_1, g_2 in q **do**
      if g_1 and g_2 unify
      then $PR := PR \cup \{\tau(\mathit{reduce}(q, g_1, g_2))\}$;
until $PR' = PR$;
return PR

Figure 10: The algorithm PerfectRef

Example 62. *Consider the query*

$$q(x)\, TeachesTo(x, y),\, TeachesTo(\_, y)$$

over the TBox of Example 33. In such a query, the atoms $TeachesTo(x, y)$ and $TeachesTo(\_, y)$ unify, and by executing reduce$(q,\, TeachesTo(x, y),\, TeachesTo(_, y))$, we obtain the atom $TeachesTo(x, y)$. The variable y is unbound, and therefore the function τ replaces it with $\_$. Now, the PI $Professor \sqsubseteq \exists TeachesTo$ can be applied to $TeachesTo(x, \_)$, whereas, before the reduction process, it could not be applied to any atom of the query. ∎

It is not difficult to show that the algorithm PerfectRef terminates, when applied to a conjunctive query and a *DL-Lite*$_{\mathcal{R}}$ TBox.

In the following we give a couple of examples of application of the algorithm PerfectRef.

Example 63. *Referring to the DL-Lite*$_{core}$ *TBox $\mathcal{T}$ in Example 33, consider the conjunctive query q:*

$$q(x)\, TeachesTo(x, y),\, HasTutor(y, \_)$$

asking for professors that teach to students that have a tutor.

Let us analyze the execution of the algorithm PerfectRef$(q, \mathcal{T})$. At the first execution of step (a), the algorithm inserts in PR the new query

$$q(x)\, TeachesTo(x, y),\, Student(y)$$

by applying to the atom $HasTutor(y, \_)$ the PI $Student \sqsubseteq \exists HasTutor$. Then, at a second execution of step (a), the query

$$q(x)\, TeachesTo(x, y),\, TeachesTo(\_, y)$$

is added to PR, according to application of the PI $\exists TeachesTo^- \sqsubseteq Student$ to the atom $Student(y)$. Since the two atoms of the second query unify, step (b) of the algorithm inserts the query

$$q(x)\, TeachesTo(x, \_)$$

Algorithm Answer$(Q, \mathcal{K})$
Input: UCQ Q, KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: $ans(Q, \mathcal{K})$
if not Consistent$(\mathcal{K})$
then return *AllTup*$(Q, \mathcal{K})$
else return $(\bigcup_{q_i \in Q} \mathsf{PerfectRef}(q_i, \mathcal{T}))^{db(\mathcal{A})}$;

Figure 11: The algorithm Answer

into PR. Notice that the variable y is unbound in the new query, hence it has been replaced by the symbol $\_$. At a next iteration, step (a) produces the query

$$q(x)Professor(x)$$

by applying $Professor \sqsubseteq \exists TeachesTo$ to $TeachesTo(x, \_)$, and then, at a further execution of step (a), it generates the query

$$q(x)HasTutor(\_, x)$$

by applying $\exists HasTutor^- \sqsubseteq Professor$ to $Professor(x)$. The set constituted by the above five queries and the original query q is then returned by the algorithm. ∎

Example 64. *As a further example, consider now the DL-Lite$_\mathcal{R}$ TBox $\mathcal{T}'$ obtained from $\mathcal{T}$ by adding the inclusion assertion $HasTutor^- \sqsubseteq TeachesTo$, and the conjunctive query q' defined as follows:*

$$q'(x)Student(x)$$

Then, the result of PerfectRef$(q', \mathcal{T}')$ *is the union of:*

$$q'(x)Student(x)$$
$$q'(x)\,TeachesTo(\_, x)$$
$$q'(x)HasTutor(x, \_)$$

Notice that, without considering the new inclusion assertion between roles, we would have obtained only the union of the first two conjunctive queries as result of the algorithm PerfectRef$(q', \mathcal{T})$. ∎

We note that the union of conjunctive queries produced by PerfectRef is not necessarily minimal, i.e., it may contain pairs of conjunctive queries that are one contained into the other. Though this does not affect the worst-case computational complexity, for practical purposes this set of queries can be simplified, using well-known minimization techniques for relational queries.

7.4.2 Query Evaluation

In order to compute the answers to q over the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we need to evaluate the set of conjunctive queries PR produced by the algorithm PerfectRef over the ABox $\mathcal{A}$ considered as a relational database.

In Figure 11, we define the algorithm Answer that, given a KB $\mathcal{K}$ and a union of conjunctive queries Q of arity n, computes $ans(Q, \mathcal{K})$.

The following theorem easily follows from termination of both the algorithm Consistent and the algorithm PerfectRef.

Theorem 65. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ KB, let Q be a union of conjunctive queries. Then, the algorithm* Answer$(Q, \mathcal{K})$ *terminates.*

Example 66. *Let us consider again the query of Example 63*

$$q(x)\, TeachesTo(x, y), HasTutor(y, \_)$$

expressed over the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{A}$ contains the assertions:

$$Student(\mathsf{John}), \quad HasTutor(\mathsf{John}, \mathsf{Mary}), \quad TeachesTo(\mathsf{Mary}, \mathsf{Bill}).$$

By executing Answer$(q, \mathcal{K})$*, since $\mathcal{K}$ is satisfiable (see Section 7.2), it executes* PerfectRef$(q, \mathcal{T})$*, which returns the union of conjunctive queries described in Example 63. Let Q be such a query, then it is easy to see that $Q^{db(\mathcal{A})}$ is the set* $\{\mathsf{Mary}\}$.

Let us now consider the query

$$q'(x) Student(x),$$

expressed over the KB $\mathcal{K}' = \langle \mathcal{T}', \mathcal{A}' \rangle$, where $\mathcal{T}'$ is as in Example 64, and $\mathcal{A}'$ contains the assertions

$$HasTutor(\mathsf{John}, \mathsf{Mary}), \quad TeachesTo(\mathsf{Mary}, \mathsf{Bill}).$$

Obviously, $\mathcal{K}'$ is satisfiable, and by executing Answer$(q', \mathcal{K}')$ *we obtain the answer set* $\{\mathsf{John}, \mathsf{Bill}\}$ *by the evaluation of the union of conjunctive queries returned by* PerfectRef$(q', \mathcal{T}')$*, and which we have described in Example 64. Notice that, without considering the new inclusion assertion between roles, we would have obtained only* $\{\mathsf{Bill}\}$ *as answer to the query.* ∎

7.4.3 Correctness

We now prove correctness of the above described query answering technique. As discussed in the previous subsection, from Theorem 58 it follows that query answering can in principle be done by evaluating the query over the model $can(\mathcal{K})$. However, since $can(\mathcal{K})$ is in general infinite, we obviously avoid the construction of $can(\mathcal{K})$. Rather, as we said before, we are able to compile the TBox into the query, thus simulating the evaluation of the query over $can(\mathcal{K})$ by evaluating a finite reformulation of the query over the ABox considered as a database.

By induction on the construction of the chase, and by induction on application of rules of the algorithm PerfectRef it is possible to show that given a *DL-Lite*$_\mathcal{R}$ TBox $\mathcal{T}$ and a conjunctive query q over $\mathcal{T}$, then for every *DL-Lite*$_\mathcal{R}$ ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that $ans(q, \langle \mathcal{T}, \mathcal{A} \rangle) = PR^{db(\mathcal{A})}$, where PR is the union of conjunctive queries returned by PerfectRef$(q, \mathcal{T})$.

Based on the above property, we are finally able to establish correctness of the algorithm Answer.

Theorem 67. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{R}$ KB, Q a union of conjunctive queries, and $\vec{t}$ a tuple of constants in $\mathcal{K}$. Then, $\vec{t} \in ans(Q, \mathcal{K})$ if and only if $\vec{t} \in$* Answer$(Q, \mathcal{K})$.

proof (sketch) In the case where $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, the proof follows immediately from the property that $ans(q, \langle \mathcal{T}, \mathcal{A} \rangle) = PR^{db(\mathcal{A})}$ and from Theorem 60. In the case where $\mathcal{K}$ is not satisfiable, it is immediate to verify that the set *AllTup*$(Q, \mathcal{K})$ returned by Answer$(Q, \mathcal{K})$ corresponds to $ans(Q, \mathcal{K})$, according to the semantics of queries given in Section 7.1.

As an immediate corollary of the above properties, it follows that the problem of answering unions of conjunctive queries over satisfiable *DL-Lite*$_\mathcal{R}$ KBs is FOL-reducible. Moreover, it is easy to see that such a FOL-reducibility also extends to the case of arbitrary (both satisfiable and unsatisfiable) *DL-Lite*$_\mathcal{R}$ KBs. Indeed, the whole query answering task can be encoded into a single union of conjunctive queries, obtained by adding to the query $\bigcup_{q_i \in Q}$ PerfectRef$(q_i, \mathcal{T}))$ a finite number of conjunctions encoding the fact that every tuple in *AllTup*$(Q, \mathcal{K})$ is in the answer set of the query if $\mathcal{K}$ is unsatisfiable. (For details on the construction of such a query see e.g. [28], which defines an analogous encoding in the context of relational database integrity constraints.) We therefore get the following theorem.

Theorem 68. *Answering unions of conjunctive queries in DL-Lite$_\mathcal{R}$ is FOL-reducible.*

7.4.4 Computational Complexity

We first notice that, for a given *DL-Lite*$_\mathcal{R}$ TBox $\mathcal{T}$, and conjunctive query q over $\mathcal{T}$, the algorithm PerfectRef $(q, \mathcal{T})$ runs in time polynomial in the size of $\mathcal{T}$.

Based on the above property, we are able to establish the complexity of answering unions of conjunctive queries in *DL-Lite*$_\mathcal{R}$.

Theorem 69. *Answering unions of conjunctive queries in DL-Lite*$_\mathcal{R}$ *is* PTIME *in the size of the TBox, and* LOGSPACE *in the size of the ABox (data complexity).*

proof (sketch) In the proof we make use of the results provided by Theorem 67 and Theorem 56, of the polynomial time complexity of the algorithm PerfectRef, and of the fact that the evaluation of a union of conjunctive queries over a database can be computed in LOGSPACE with respect to the size of the database (since unions of conjunctive queries are a subclass of FOL queries).

We are also able to characterize the combined complexity (i.e., the complexity w.r.t. the size of $\mathcal{K}$ and Q) of answering unions of conjunctive queries in *DL-Lite*$_\mathcal{R}$.

Theorem 70. *Answering unions of conjunctive queries in DL-Lite*$_\mathcal{R}$ *is NP-complete in combined complexity.*

Proof. To prove membership in NP, observe that a version of the algorithm PerfectRef that nondeterministically returns only one of the conjunctive queries belonging to the reformulation of the input query, runs in nondeterministic polynomial time in combined complexity, since every query returned by PerfectRef can be generated after a polynomial number of transformations of the initial conjunctive query (i.e., after a polynomial number of executions of steps (a) and (b) of the algorithm). This allows the corresponding nondeterministic version of the algorithm Answer to run in nondeterministic polynomial time when the input is a boolean query. NP-hardness follows from NP-hardness of conjunctive query evaluation over relational databases. ❑

Summarizing, the above results show a very nice computational behavior of queries in *DL-Lite*$_\mathcal{R}$: reasoning in *DL-Lite*$_\mathcal{R}$ is computationally no worse than standard conjunctive query answering (and containment) in relational databases.

7.5 Query Answering in *DL-Lite*$_\mathcal{F}$

In this subsection, we discuss query answering in *DL-Lite*$_\mathcal{F}$, and analyze its computational complexity. In a nutshell, the technique for query answering closely resembles that for *DL-Lite*$_\mathcal{R}$, hence, it is also based on reformulating the query based on the TBox assertions. The differences with respect to the case of *DL-Lite*$_\mathcal{R}$ are the following. (*i*) On the one hand, the PIs that can appear in a *DL-Lite*$_\mathcal{F}$ TBox are just a subset of those allowed for a *DL-Lite*$_\mathcal{R}$ TBox, namely inclusions of a basic concept in a concept, but no role inclusions. As a consequence, the reformulation rules that can be applied to the atoms of the query are just a subset of those for *DL-Lite*$_\mathcal{R}$ in Definition 61. (*ii*) On the other hand, a *DL-Lite*$_\mathcal{F}$ TBox may contain functionality assertions, which may interact with the inclusion assertions in the TBox. However, this interaction is only of a limited form. Indeed, as already shown in Lemma 40, if a functionality assertion $(\mathsf{funct}\ R)$ is satisfied in the interpretation $db(\mathcal{A})$ corresponding to the ABox $\mathcal{A}$ of a KB $\mathcal{K}$, then $(\mathsf{funct}\ R)$ is also satisfied in the canonical interpretation $can(\mathcal{K})$. Hence, (the simplified form of) the reformulation technique of *DL-Lite*$_\mathcal{R}$ can also be applied to *DL-Lite*$_\mathcal{F}$ query answering, provided that we take into account also functionality assertions when we check satisfiability of the knowledge base. We now show this formally.

Given a *DL-Lite*$_\mathcal{F}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a conjunctive query q, the reformulation PR of q is computed, just as for *DL-Lite*$_\mathcal{R}$, by the algorithm PerfectRef, considering that inclusion assertions between roles

are not present in $\mathcal{T}$, and hence the rules in Definition 61 for such inclusions assertions will never be applied.

Similarly, in order to compute the answer $ans(Q, \mathcal{K})$ to the union of conjunctive queries Q over a *DL-Lite*$_{\mathcal{F}}$ KB $\mathcal{K}$, we can simply invoke Answer on Q and $\mathcal{K}$, noticing that now the satisfiability check takes into account also functionality assertions in $\mathcal{K}$. It is easy to see that also in the case when $\mathcal{K}$ is a *DL-Lite*$_{\mathcal{F}}$ KB the algorithm Answer$(Q, \mathcal{K})$ terminates (cf. Theorem 65).

The following theorem, which follows easily from the already proved results, establishes that this way of proceeding is indeed correct.

Theorem 71. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{F}}$ KB, Q a union of conjunctive queries, and $\vec{t}$ a tuple of constants in $\mathcal{K}$. Then, $\vec{t} \in ans(Q, \mathcal{K})$ if and only if $\vec{t} \in$ Answer$(Q, \mathcal{K})$.*

As an immediate consequence of the theorem above, we get the following property, which is the analogous of Theorem 68, given for *DL-Lite*$_{\mathcal{R}}$.

Theorem 72. *Answering unions of conjunctive queries in DL-Lite$_{\mathcal{F}}$ is FOL-reducible.*

As for computational complexity we get the same bounds as those shown in Section 7.4.

Theorem 73. *Answering unions of conjunctive queries in DL-Lite$_{\mathcal{F}}$ is* PTIME *in the size of the TBox,* LOGSPACE *in the size of the ABox (data complexity), and NP-complete in combined complexity.*

8 Updating Description Logic ABoxes

8.1 Introduction

Description logic *ABoxes* have the purpose to describe a snapshot of the world. For example, the following ABox, which is formulated in $\mathcal{ALC}$, says that John is a parent with only happy children, that Peter is his child, and that Mary is a person:

$$\begin{aligned}&\mathsf{john{:}Person} \sqcap \exists \mathsf{has\text{-}child.Person} \sqcap \\ &\qquad\qquad \forall \mathsf{has\text{-}child.}(\mathsf{Person} \sqcap \mathsf{Happy}) \\ &\mathsf{has\text{-}child}(\mathsf{john}, \mathsf{peter}) \\ &\mathsf{mary{:}Person}\end{aligned}$$

In many applications of DLs, an ABox is used to represent the current state of affairs in the application domain [8]. In such applications, it is necessary to update the ABox in the case that the state has changed. Such an update should incorporate the information about the new state while retaining all knowledge that is not affected by the change (as demanded by the principle of inertia, see e.g. [114]). For example, if Mary is now unhappy, we should update the above ABox to the following one. This updated ABox is formulated in $\mathcal{ALCO}$, the extension of $\mathcal{ALC}$ with nominals (i.e., individual names inside concept descriptions):

$$\begin{aligned}&\mathsf{john{:}Person} \sqcap \exists \mathsf{has\text{-}child.Person} \sqcap \\ &\qquad\qquad \forall \mathsf{has\text{-}child.}(\mathsf{Person} \sqcap (\mathsf{Happy} \sqcup \{\mathsf{mary}\})) \\ &\mathsf{has\text{-}child}(\mathsf{john}, \mathsf{peter}) \\ &\mathsf{mary{:}Person} \sqcap \neg \mathsf{Happy}\end{aligned}$$

To see that this ABox is obtained by the update operation, note that ABoxes adopt the open world assumption and thus represent the domain in an incomplete way [8]. In the example above, we have no information about whether or not Mary is a child of John. However, because we cannot exclude that this is the case, John may now have an unhappy child (which is Mary). Thus, the new information concerning Mary also resulted in an update of the information concerning John.

In applications, ABoxes are usually updated in an ad-hoc way, and effects such as the information change for John above are simply ignored. The purpose of the current section is *to provide a formal analysis of ABox updates in many common description logics, concentrating on the most basic kind of updates*. The basic kind of update we consider is as follows: the new information to be incorporated into the ABox is a set of possibly negated assertions $a{:}A$ and $r(a, b)$, where A is an *atomic* concept. As discussed in more detail later, there are both semantic and computational problems with unrestricted updates that are avoided by adopting these restrictions.

We consider ABox updates in the expressive DL $\mathcal{ALCQIO}$ and its fragments. It turns out that, in many natural DLs such as $\mathcal{ALC}$, the updated ABox cannot be expressed. As an example, consider the $\mathcal{ALC}$ ABox given above. To express the ABox obtained by the update with $\mathsf{mary{:}}\neg\mathsf{Happy}$, we had to resort to the more expressive DL $\mathcal{ALCO}$. But even the introduction of nominals does not suffice to guarantee that updated ABoxes are expressible. Only if we further add the "@" concept constructor from hybrid logic [5] or Boolean ABoxes (we show that these two are equivalent in the presence of nominals), updated ABoxes can be expressed. Here, the @ constructor allows the formation of *concepts* of the form $@_a C$ expressing that the individual a satisfies C, and Boolean ABoxes are a generalization of standard ABoxes: while the latter can be thought of as a conjunction of ABox assertions of the form $a{:}C$ and $r(a, b)$, Boolean ABoxes are a Boolean combination of such ABox assertions. Our expressiveness

results do not only concern $\mathcal{ALC}$: similar proofs as those given in this section can be used to show that, in any standard DL in which nominals and the "@" constructor are not expressible, updated ABoxes cannot be expressed.

We show that updated ABoxes exist and are computable in $\mathcal{ALCQIO}^{@}$, the extension of $\mathcal{ALCQIO}$ (which includes nominals) with the @ constructor. The proposed algorithm can easily be adapted to the fragments $\mathcal{ALCO}^{@}$, $\mathcal{ALCIO}^{@}$, and $\mathcal{ALCQO}^{@}$. An important issue is the size of updated ABoxes: the updated ABoxes computed by our algorithm may be of size exponential both in the size of the original ABox and in the size of the new information (henceforth called the *update*). We show that an exponential blowup cannot be completely avoided by proving that, even in the case of propositional logic, the size of updated theories is not polynomial in the size of the (combined) input unless every PTIME-algorithm is LOGTIME-parallelizable (the "P vs. $\mathsf{N_P}$" question of parallel computation).[24] In the update literature, an exponential blowup in the size of the update is viewed as much more tolerable than an exponential blowup in the size of the original ABox since the former tend to be small compared to the latter. We believe that, in the case of $\mathcal{ALCQIO}^{@}$ and its two fragments mentioned above, the exponential blowup in the size of the original ABox cannot be avoided. While we leave a proof as an open problem, we exhibit two ways around the blowup: the first is to allow the introduction of new concept definitions in an acyclic TBox when computing the update. The second is to move to extensions of $\mathcal{ALCQIO}^{@}$ that allow Boolean operators on roles, thus eliminating the asymmetry between concepts and roles found in standard DLs. In both cases, we show how to compute updated ABoxes that are polynomial in the size of the original ABox (and exponential in the size of the update). Thus, the blowup induced by updates in these expressive DLs is not worse than in propositional logic. We also show that the blowup produced by iterated updates is not worse than the blowup produced by a single update.

The results presented in this section are published as a part of [100] and the full proofs can be found in the accompanying technical report [101].

8.2 Description Logic $\mathcal{ALCQIO}^{@}$

In DLs, *concepts* are inductively defined with the help of a set of *constructors*, starting with a set $\mathsf{N_C}$ of *concept names* and a set $\mathsf{N_R}$ of *role names*, and (possibly) a set $\mathsf{N_I}$ of *individual names*. In this section, we introduce the DL $\mathcal{ALCQIO}^{@}$, whose concepts are formed using the constructors shown in Figure 12.

There, the inverse constructor is the only role constructor, whereas the remaining seven constructors are concept constructors. In Figure 12 and in what follows, we use $\#S$ to denote the cardinality of a set S, a and b to denote individual names, r and s to denote roles (i.e., role names and inverses thereof), A, B to denote concept names, and C, D to denote (possibly complex) concepts. As usual, we use $\top$ as abbreviation for an arbitrary (but fixed) propositional tautology, $\bot$ for $\neg\top$, $\rightarrow$ and $\leftrightarrow$ for the usual Boolean abbreviations, $\exists r.C$ (*existential restriction*) for $(\geqslant 1\ r\ C)$, and $\forall r.C$ (*universal restriction*) for $(\leqslant 0\ r\ \neg C)$.

The DL that allows only for negation, conjunction, disjunction, and universal and existential restrictions is called $\mathcal{ALC}$. The availability of additional constructors is indicated by concatenation of a corresponding letter: $\mathcal{Q}$ stands for number restrictions; $\mathcal{I}$ stands for inverse roles, $\mathcal{O}$ for nominals and superscript "@" for the @ constructor. This explains the name $\mathcal{ALCQIO}^{@}$ for our DL, and also allows us to refer to its sublanguages in a simple way.

[24] In contrast to the results by Cadoli et al. [27], our result even applies to the restricted form of updates, i.e., updates in propositional logic where the update is a conjunction of literals. Thus, our argument provides further evidence for the claims in [27], where it is shown that, with unrestricted updates, an exponential blowup in the size of the update cannot be avoided unless the first levels of the polynomial hierarchy collapse.

Name	Syntax	Semantics
inverse role	r^-	$(r^{\mathcal{I}})^{-1}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
at-least restriction	$(\geqslant n\ r\ C)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$
at-most restriction	$(\leqslant n\ r\ C)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$
@ constructor	$@_a C$	$\Delta^{\mathcal{I}}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $\emptyset$ otherwise

Figure 12: Syntax and semantics of $\mathcal{ALCQIO}^{@}$.

The semantics of $\mathcal{ALCQIO}^{@}$-concepts is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set of individuals and the *interpretation function* $\cdot^{\mathcal{I}}$ maps each concept name $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name $r \in \mathsf{N_R}$ to a binary relation $r^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, and each individual name $a \in \mathsf{N_I}$ to an individual $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to inverse roles and arbitrary concepts is inductively defined as shown in the third column of Figure 12.

An $\mathcal{ALCQIO}^{@}$ *assertional box (ABox)* is a finite set of *concept assertions* $C(a)$ and *role assertions* $r(a,b)$ and $\neg r(a,b)$ (where r may be an inverse role). For readability, we sometimes write concept assertions as $a{:}C$. Observe that there is no need for explicitly introducing negated concept assertions as negation is available as a concept constructor in $\mathcal{ALCQIO}^{@}$. An ABox $\mathcal{A}$ is *simple* if $C(a) \in \mathcal{A}$ implies that C is a *concept literal*, i.e., a concept name or a negated concept name.

An interpretation $\mathcal{I}$ *satisfies* a concept assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, a role assertion $r(a,b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, and a role assertion $\neg r(a,b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$. We denote satisfaction of an ABox assertion α by an intepretation $\mathcal{I}$ with $\mathcal{I} \models \alpha$. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ (written $\mathcal{I} \models \mathcal{A}$) if it satisfies all assertions in $\mathcal{A}$. An ABox is *consistent* iff it has a model. Two ABoxes $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* (written $\mathcal{A} \equiv \mathcal{A}'$) iff they have the same models. We use $M(\mathcal{A})$ to denote the set of all models of the ABox $\mathcal{A}$.

Various reasoning problems are considered for DLs. For the purpose of this paper, it suffices to introduce concept satisfiability and ABox consistency:

- the concept C is *satisfiable* w.r.t. the TBox $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$;
- the ABox $\mathcal{A}$ is *consistent* w.r.t. the TBox $\mathcal{T}$ iff there exists an interpretation $\mathcal{I}$ that is a model of both $\mathcal{T}$ and $\mathcal{A}$.

8.3 ABox Updates

We introduce a simple form of ABox update where complex concepts are not allowed in the update information.

Definition 74 (Interpretation Update)**.** An *update* $\mathcal{U}$ is a simple ABox that is consistent. Let $\mathcal{U}$ be an update and $\mathcal{I}$, $\mathcal{I}'$ interpretations such that $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}'}$ and $\mathcal{I}$ and $\mathcal{I}'$ agree on the interpretation of

individual names. Then $\mathcal{I}'$ is the *result of updating* $\mathcal{I}$ *with* $\mathcal{U}$*, written* $\mathcal{I} \Longrightarrow_{\mathcal{U}} \mathcal{I}'$, if the following hold for all concept names A and role names r:

$$\begin{aligned} A^{\mathcal{I}'} &= (A^{\mathcal{I}} \cup \{a^{\mathcal{I}} \mid A(a) \in \mathcal{U}\}) \setminus \{a^{\mathcal{I}} \mid \neg A(a) \in \mathcal{U}\} \\ r^{\mathcal{I}'} &= (r^{\mathcal{I}} \cup \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid r(a,b) \in \mathcal{U}\}) \\ &\qquad \setminus \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \neg r(a,b) \in \mathcal{U}\} \end{aligned}$$

It is easily seen that, for each fixed update $\mathcal{U}$, the relation "$\Longrightarrow_{\mathcal{U}}$" is functional. Therefore, we can write $\mathcal{I}^{\mathcal{U}}$ to denote the unique $\mathcal{I}'$ with $\mathcal{I} \Longrightarrow_{\mathcal{U}} \mathcal{I}'$. △

Based on the relation "$\Longrightarrow_{\mathcal{U}}$", we can now define ABox updates.

Definition 75 (ABox Update)**.** Let $\mathcal{A}$ be an ABox and $\mathcal{U}$ an update. An ABox $\mathcal{A}'$ is the *result of updating* $\mathcal{A}$ *with* $\mathcal{U}$, in symbols $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$, if

$$M(\mathcal{A}') = \{\mathcal{I}^{\mathcal{U}} \mid \mathcal{I} \in M(\mathcal{A})\}.$$

We then call $\mathcal{A}$ the *original ABox* and $\mathcal{A}'$ the *updated ABox*. △

Note that updated ABoxes are unique up to logical equivalence. This justifies talking of *the* result of updating $\mathcal{A}$ with $\mathcal{U}$.

There are at least two advantages of disallowing complex concepts in updates: first, the semantics given above is uncontroversial and coincides with all standard semantics for updates considered in the literature, see e.g., [142, 133, 160, 128]. In contrast, several different and equally natural semantics become available as soon as we allow disjunction (or even non-Boolean constructors) in updates, see e.g. [160, 51, 99, 141, 161, 71, 128]. Second, it follows from the results in [10] that, at least under Winslett-style PMA semantics [160], unrestricted ABox updates in relatively simple DLs are not computable. It seems very likely that the other available semantics suffer from similar computational problems. Practically, our restriction means that the user has to describe updates at an atomic level.

We now give a more involved example of updating ABoxes than that given in the introduction. The following $\mathcal{ALCO}$ ABox expresses that John and Mary are married. We also know that (at least) one of them is unhappy, but we do not know which of the two this is. Moreover, Peter and Sarah both have a happy parent:

$$\mathsf{spouse}(\mathsf{john}, \mathsf{mary})$$

$$\mathsf{peter}{:}\exists \mathsf{parent}.\mathsf{Happy}$$

$$\mathsf{sarah}{:}\exists \mathsf{parent}.\mathsf{Happy}$$

$$\mathsf{john}{:}\neg \mathsf{Happy} \sqcup \forall \mathsf{spouse}.(\{\mathsf{mary}\} \rightarrow \neg \mathsf{Happy})$$

Suppose that, because one of them is unhappy, John and Mary have an argument. This results in both John and Mary being unhappy now. Hence, we should apply the following update to the above ABox:

$$\neg \mathsf{Happy}(\mathsf{john}), \quad \neg \mathsf{Happy}(\mathsf{mary}).$$

Then, the updated ABox can be expressed in $\mathcal{ALCO}^{@}$ as follows. Here and in what follows, we assume

that the @ constructor has higher precedence than conjunction:

$$\begin{aligned}
&\mathsf{spouse}(\mathsf{john}, \mathsf{mary})\\
&\mathsf{dummy}{:}\big(@_{\mathsf{peter}}\exists\mathsf{parent}.(\mathsf{Happy} \sqcup \{\mathsf{john}\})\sqcap\\
&\qquad @_{\mathsf{sarah}}\exists\mathsf{parent}.(\mathsf{Happy} \sqcup \{\mathsf{john}\})\big)\sqcup\\
&\qquad \big(@_{\mathsf{peter}}\exists\mathsf{parent}.(\mathsf{Happy} \sqcup \{\mathsf{mary}\})\sqcap\\
&\qquad @_{\mathsf{sarah}}\exists\mathsf{parent}.(\mathsf{Happy} \sqcup \{\mathsf{mary}\})\big)\\
&\neg\mathsf{Happy}(\mathsf{john})\\
&\neg\mathsf{Happy}(\mathsf{mary})
\end{aligned}$$

The only surprising assertion in the updated ABox is the second one. Intuitively, it represents the update of the second and third assertion of the original ABox:[25] the first disjunct captures the case where John was unhappy and Mary was happy, and the second disjunct captures the case when it was the other way around. In the remaining that both Mary *and* John were unhappy, the update of the second and third assertions is

$$\mathsf{dummy}{:}@_{\mathsf{peter}}\exists\mathsf{parent}.\mathsf{Happy} \sqcap @_{\mathsf{sarah}}\exists\mathsf{parent}.\mathsf{Happy}$$

(because nothing has changed through the update). A corresponding disjunct in the second assertion of the updated ABox is not needed since it would imply each of the first two disjuncts.

Also note that the last line of the original ABox is subsumed by the last two lines of the updated ABox.

We shall later refer back to this example as the "spouse example" and prove that the updated ABox cannot be expressed in $\mathcal{ALCO}$.

8.4 Description Logics without Updates

We say that a description logic $\mathcal{L}$ *has ABox updates* iff, for every ABox $\mathcal{A}$ formulated in $\mathcal{L}$ and every update $\mathcal{U}$, there exists an ABox $\mathcal{A}'$ formulated in $\mathcal{L}$ such that $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$. In this section, we show that a lot of basic DLs are lacking ABox updates.

8.4.1 Updates in $\mathcal{ALC}$

We analyze the basic description logic $\mathcal{ALC}$ and show that it does not have ABox updates. Consider the following $\mathcal{ALC}$ ABox $\mathcal{A}$, update $\mathcal{U}$, and $\mathcal{ALCO}$ ABox $\mathcal{A}'$:

$$\begin{aligned}
\mathcal{A} &:= \{\forall r.A(a)\}\\
\mathcal{U} &:= \{\neg A(b)\}\\
\mathcal{A}' &:= \{\neg A(b), \forall r.(A \sqcup \{b\})(a)\}.
\end{aligned}$$

The following is not difficult to verify using Definition 75.

Lemma 76. $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$.

[25] Note that the individual name dummy in front of the colon does not carry any information: we could exchange it with any other individual name without changing the meaning of this assertion.

To show that $\mathcal{ALC}$ does not have ABox updates, it thus suffices to prove that there is no $\mathcal{ALC}$ ABox equivalent to the $\mathcal{ALCO}$ ABox $\mathcal{A}'$. This is an easy exercise: find two models $\mathcal{I}$ and $\mathcal{I}'$ that are bisimilar[26] such that $\mathcal{I} \models \mathcal{A}'$ and $\mathcal{I}' \not\models \mathcal{A}'$. Then use the fact that $\mathcal{ALC}$ concepts cannot distinguish between bisimilar models to show the desired result. Details of this and following proofs can be found in [101].

Lemma 77. *There is no $\mathcal{ALC}$ ABox equivalent to $\mathcal{A}'$.*

Note that our proof applies to the case where the update contains only concept assertions, but no role assertions.

Theorem 78. *$\mathcal{ALC}$ does not have ABox updates, even if updates contain only concept assertions.*

The fact that the updated ABox $\mathcal{A}'$ used in this section is actually an $\mathcal{ALCO}$ ABox may give rise to the conjecture that adding nominals to $\mathcal{ALC}$ recovers the existence of updates. Unfortunately, as shown in the following section, this is not the case.

8.4.2 Updates in $\mathcal{ALCO}$

To show that $\mathcal{ALCO}$ does not have ABox updates, we proceed in two steps: we first give a straightforward proof of the non-existence of updated ABoxes in $\mathcal{ALCO}$ that relies on the use of role assertions in updates. In the second step, we use a slightly more complex construction to show that $\mathcal{ALCO}$ does not have ABox updates even if only concept assertions are allowed in updates.

Consider the following $\mathcal{ALC}$ ABox $\mathcal{A}$ (which thus also is an $\mathcal{ALCO}$ ABox), update $\mathcal{U}$, and $\mathcal{ALCO}^{@}$ ABox $\mathcal{A}'$:

$$\begin{aligned} \mathcal{A} &:= \{\exists r.A(a)\} \\ \mathcal{U} &:= \{\neg r(a,b)\} \\ \mathcal{A}' &= \{(\exists r.(A \sqcap \neg\{b\}) \sqcup @_b A)(a), \neg r(a,b)\}. \end{aligned}$$

Again, the following is not difficult to verify:

Lemma 79. $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$.

We now show that there exists no $\mathcal{ALCO}$ ABox that is equivalent to the $\mathcal{ALCO}^{@}$ ABox $\mathcal{A}'$. It follows that $\mathcal{ALCO}$ does not have ABox updates.

Consider the interpretations $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}''$ depicted in Figure 13. We assume that the individual names a, b, and c are mapped to the individuals of the same name, and that all other individual names are mapped to the individual c. Moreover, the concept name A is interpreted as shown in the figure and all other concept names are interpreted as the empty set in all three interpretations. Then we have $\mathcal{I} \models \mathcal{A}'$, $\mathcal{I}' \models \mathcal{A}'$ and $\mathcal{I}'' \not\models \mathcal{A}'$. We will show that, if an $\mathcal{ALCO}$ ABox $\mathcal{B}$ is equivalent to $\mathcal{A}'$, then $\mathcal{I}'' \models \mathcal{B}$, which is a contradiction.

Lemma 80. *There is no $\mathcal{ALCO}$ ABox that is equivalent to the $\mathcal{ALCO}^{@}$ ABox $\mathcal{A}' = \{(\exists r.(A \sqcap \neg\{b\}) \sqcup @_b A)(a), \neg r(a,b)\}$.*

Proof. Assume there is an $\mathcal{ALCO}$ ABox $\mathcal{B}$ that is equivalent to $\mathcal{A}'$. Then $\mathcal{I} \models \mathcal{B}$, $\mathcal{I}' \models \mathcal{B}$, and $\mathcal{I}'' \not\models \mathcal{B}$. We show that, for all assertions $\varphi \in \mathcal{B}$, we have $\mathcal{I}'' \models \varphi$, thus obtaining a contradiction to $\mathcal{I}'' \not\models \mathcal{B}$. First, $\mathcal{B}$ does not contain any positive role assertion since $\mathcal{I} \models \mathcal{B}$ and $\mathcal{I}$ does not satisfy any positive role assertions. Second, if φ is a negative role assertion, then $\mathcal{I}'' \models \varphi$ since $\mathcal{I}''$ satisfies all negative role assertions. Finally, let φ be a concept assertion. Then, $\mathcal{I}'' \models \varphi$ is a consequence of $\mathcal{I} \models \varphi$, $\mathcal{I}' \models \varphi$, and the fact that the truth of an assertion $C(\hat{a})$ in a model $\mathcal{J}$, C an $\mathcal{ALCO}$-concept, only depends on the set of points reachable from $\hat{a}^{\mathcal{J}}$ by role paths. ❏

[26] W.r.t. the standard notion of bisimilarity for $\mathcal{ALC}$ that is independent of individual names [93].

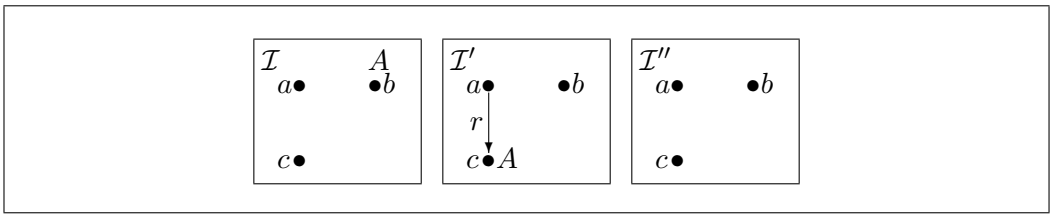

Figure 13: Interpretations for Lemma 80

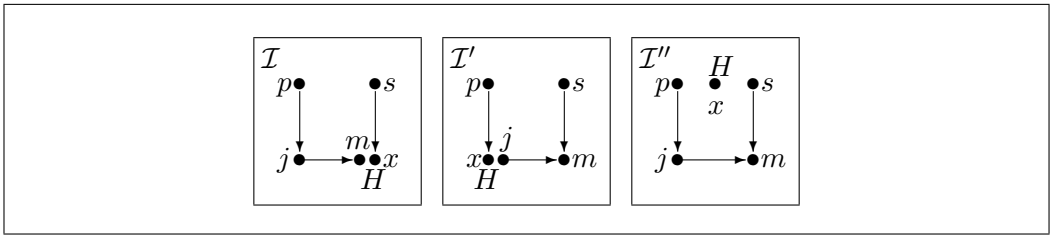

Figure 14: Interpretations for Lemma 82

Note that our proof also shows that $\mathcal{ALC}$ does not have ABox updates even if we restrict ourselves to updates containing only role assertions.

Theorem 81. *$\mathcal{ALC}$ and $\mathcal{ALCO}$ do not have ABox updates, even if updates contain only role assertions.*

This result raises the question whether or not restricting updates to concept assertions regains the existence of updated ABoxes in $\mathcal{ALCO}$. We answer this question to the negative by returning to the spouse example. Let $\mathcal{A}, \mathcal{U}$, and $\mathcal{A}'$ denote the original ABox (formulated in $\mathcal{ALCO}$), update, and updated ABox (formulated in $\mathcal{ALCO}^{@}$) from this example. We have already argued that $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$. It suffices to prove that there is no $\mathcal{ALCO}$ ABox equivalent to $\mathcal{A}'$.

Consider the interpretations $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}''$ depicted in Figure 14 where we abbreviate the individual names from the spouse example using their initial letter, x denotes an individual, all horizontal edges are for the role spouse, and all vertical edges are for the role parent. We assume that the four individual names j, m, p, s are mapped to individuals of the same name, and that all other individual names are mapped to the individual x. Moreover, H is interpreted as indicated and all other concept names are interpreted as the empty set.

The proof of the following lemma uses the facts that $\mathcal{I} \models \mathcal{A}'$, $\mathcal{I}' \models \mathcal{A}'$, but $\mathcal{I}'' \not\models \mathcal{A}'$. It is quite similar to the proof of Lemma 80.

Lemma 82. *There is no $\mathcal{ALCO}$-ABox that is equivalent to the updated ABox from the spouse example.*

Thus, we obtain the following result:

Theorem 83. *$\mathcal{ALCO}$ does not have ABox updates, even if updates contain only concept assertions.*

8.4.3 Updates in $\mathcal{ALC}^{@}$ and Boolean ABoxes in $\mathcal{ALC}$

The proofs of Theorems 81 and 83 suggest that there is a connection between ABox updates and the "@" constructor. Indeed, we will later see that the DL $\mathcal{ALCO}^{@}$ has ABox updates. Here, we show that adding *only* the @ constructor to $\mathcal{ALC}$ does not suffice to guarantee the existence of updated ABoxes. Indeed, we even consider Boolean ABoxes [3], which are closely related to the @ constructor but strictly more expressive.

Boolean ABox assertions are Boolean combinations of ABox assertions expressed in terms of the connectives $\wedge$ and $\vee$. Then, a *Boolean ABox* is simply a finite set of Boolean ABox assertions. We do not need to explicitly introduce negation since we admit negated role assertions and concept negation is contained in every DL considered in this section. For example, the following is a Boolean ABox:

$$\{B(a), (A(a) \wedge r(a,b)) \vee \neg\exists s.A(b)\}.$$

An interpretation $\mathcal{I}$ is a model of a Boolean ABox $\mathcal{A}$ if every Boolean ABox assertion in $\mathcal{A}$ evaluates to true. The following lemma relates Boolean ABoxes and the @ constructor. It shows that non-Boolean $\mathcal{ALCO}^{@}$ ABoxes have exactly the same expressive power as Boolean $\mathcal{ALCO}$-ABoxes, and that the same does not hold for $\mathcal{ALC}$: while every $\mathcal{ALC}^{@}$ ABox can be translated into an equivalent Boolean $\mathcal{ALC}$ ABox, there are Boolean $\mathcal{ALC}$ ABoxes for which no equivalent non-Boolean $\mathcal{ALC}^{@}$ ABox exists.

Lemma 84.
(i) For every Boolean $\mathcal{ALC}^{@}$ ABox ($\mathcal{ALCO}^{@}$ ABox), there exists an equivalent Boolean $\mathcal{ALC}$ ABox ($\mathcal{ALCO}$ ABox);
(ii) For every Boolean $\mathcal{ALCO}$ ABox, there exists an equivalent non-Boolean $\mathcal{ALCO}^{@}$ ABox;
(iii) There exists no non-Boolean $\mathcal{ALC}^{@}$ ABox that is equivalent to the Boolean $\mathcal{ALC}$ ABox $\{A(a) \vee r(b,c)\}$.

Since, when added to $\mathcal{ALC}$, Boolean ABoxes are more expressive than the @ constructor, it is more general to consider the former when proving the lack of ABox updates.

Theorem 85. *There exists an $\mathcal{ALC}$ ABox $\mathcal{A}$ and an update $\mathcal{U}$ such that there exists no Boolean $\mathcal{ALC}$ ABox $\mathcal{A}'$ with $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$.*

The proof of Theorem 85 uses the ABoxes $\mathcal{A}$, $\mathcal{A}'$ and the update $\mathcal{U}$ that have been used in the proof that $\mathcal{ALC}$ does not have ABox updates. To show that no Boolean ABox $\mathcal{B}$ is equivalent to $\mathcal{A}$, $\mathcal{B}$ is first converted into disjunctive normal form and then proceeds similar to the non-Boolean case. By Lemma 84, we obtain the following corollary.

Corollary 86. *$\mathcal{ALC}^{@}$ does not have ABox updates.*

Observe that both Theorem 85 and Corollary 86 remain true if we restrict updates to only concept assertions.

8.5 Computing Updates in $\mathcal{ALCQIO}^{@}$

Straightforward extensions of the results obtained in the previous section show that none of the standard DLs between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$ has ABox updates. In this section, we show that adding nominals and the @ constructor to such DLs suffices to have ABox updates. More presicely, we prove that the expressive DL $\mathcal{ALCQIO}^{@}$ has ABox updates. The proof is easily adapted to the fragments of $\mathcal{ALCQIO}^{@}$ obtained by dropping number restrictions, inverse roles, or both.

Our construction of updated ABoxes is an extension of the corresponding construction for propositional logic described in [160], and proceeds as follows. First, we consider *updates of concepts* on the level of interpretations. More precisely, we show how to convert a concept C and an update $\mathcal{U}$ into a concept $C^{\mathcal{U}}$ such that the following holds:

($*$) for all interpretations $\mathcal{I}$ and $\mathcal{I}'$ such that $\mathcal{I}$ satisfies *no* assertion in $\mathcal{U}$ and $\mathcal{I} \Longrightarrow_{\mathcal{U}} \mathcal{I}'$, we have $C^{\mathcal{I}} = (C^{\mathcal{U}})^{\mathcal{I}'}$.

So intuitively, $C^{\mathcal{U}}$ can be used after the update to describe exactly those objects that have been in the

$$
\begin{array}{rcl}
A^{\mathcal{U}} & = & A \sqcup \bigsqcup_{\neg A(a)\in\mathcal{U}} \{a\} \sqcap \neg(\bigsqcup_{A(a)\in\mathcal{U}} \{a\}) \qquad\qquad \{a\}^{\mathcal{U}} = \{a\} \\
(@_a C)^{\mathcal{U}} & = & @_a C^{\mathcal{U}} \qquad\qquad (\neg C)^{\mathcal{U}} = \neg C^{\mathcal{U}} \\
(C \sqcap D)^{\mathcal{U}} & = & C^{\mathcal{U}} \sqcap D^{\mathcal{U}} \qquad\qquad (C \sqcup D)^{\mathcal{U}} = C^{\mathcal{U}} \sqcup D^{\mathcal{U}} \\
(\geqslant m\ r\ C)^{\mathcal{U}} & = & (\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \sqcap (\geq m\ r\ C^{\mathcal{U}})) \\
& & \sqcup \bigsqcup_{a\in \mathsf{Obj}(\mathcal{U})} \Big(\{a\} \sqcap \bigsqcup_{m_1+m_2+m_3=m} \big((\geq m_1\ r \bigsqcap_{b\in \mathsf{Obj}(\mathcal{U})} \neg\{b\} \sqcap C^{\mathcal{U}}) \\
& & \qquad \sqcap (\geq m_2\ r \bigsqcup_{b\in \mathsf{Obj}(\mathcal{U}), r(a,b)\notin\mathcal{U}} \{b\} \sqcap C^{\mathcal{U}}) \\
& & \qquad \sqcap \bigsqcup_{S\subseteq\{b\mid \neg r(a,b)\in\mathcal{U}\}, |S|=m_3} \bigsqcap_{b\in S} @_b C^{\mathcal{U}})\Big) \\
(\leqslant m\ r\ C)^{\mathcal{U}} & = & (\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \sqcap (\leq m\ r\ C^{\mathcal{U}})) \\
& & \sqcup \bigsqcup_{a\in \mathsf{Obj}(\mathcal{U})} \Big(\{a\} \sqcap \bigsqcup_{m_1+m_2+m_3=m} \big((\leq m_1\ r \bigsqcap_{b\in \mathsf{Obj}(\mathcal{U})} \neg\{b\} \sqcap C^{\mathcal{U}}) \\
& & \qquad \sqcap (\leq m_2\ r \bigsqcup_{b\in \mathsf{Obj}(\mathcal{U}), r(a,b)\notin\mathcal{U}} \{b\} \sqcap C^{\mathcal{U}}) \\
& & \qquad \sqcap \bigsqcap_{S\subseteq\{b\mid \neg r(a,b)\in\mathcal{U}\}, |S|=m_3+1} \bigsqcup_{b\in S} \neg @_b C^{\mathcal{U}})\Big)
\end{array}
$$

Figure 15: Constructing $C^{\mathcal{U}}$

extension of C before the update. Our aim is to use the translation $C^{\mathcal{U}}$ to update concept assertions in ABoxes. We will later see how to overcome the restriction that $\mathcal{I}$ has to satisfy no assertion in $\mathcal{U}$.

For defining the concepts $C^{\mathcal{U}}$, we first introduce a bit of notation. For an ABox $\mathcal{A}$, we use $\mathsf{Obj}(\mathcal{A})$ to denote the set of individual names in $\mathcal{A}$, and $\mathsf{sub}(\mathcal{A})$ to denote the set of subconcepts of the concepts occurring in $\mathcal{A}$. For an ABox $\mathcal{A}$, we use $\neg\mathcal{A}$ to denote $\{\neg\varphi \mid \varphi \in \mathcal{A}\}$. The inductive translation that takes a concept C and an update $\mathcal{U}$ to a concept $C^{\mathcal{U}}$ as explained above is given in

Lemma 87. *The translation of concepts C into concepts $C^{\mathcal{U}}$ given in Figure 15 satisfies $(*)$.*

We now extend the update of concepts to the update of ABoxes. Let $\mathcal{A}$ be an ABox and $\mathcal{U}$ an update. Then define the ABox $\mathcal{A}^{\mathcal{U}}$ by setting

$$
\begin{array}{rcl}
\mathcal{A}^{\mathcal{U}} & := & \{C^{\mathcal{U}}(a) \mid C(a) \in \mathcal{A}\} \cup \\
& & \{r(a,b) \mid r(a,b) \in \mathcal{A} \wedge \neg r(a,b) \notin \mathcal{U}\} \cup \\
& & \{\neg r(a,b) \mid \neg r(a,b) \in \mathcal{A} \wedge r(a,b) \notin \mathcal{U}\}.
\end{array}
$$

We can now establish a property that corresponds to $(*)$, but concerns ABoxes instead of concepts.

Lemma 88. *Let $\mathcal{A}$ be an ABox and $\mathcal{U}$ an update. For every interpretation $\mathcal{I}$ with $\mathcal{I} \models \neg\mathcal{U}$, we have $\mathcal{I} \models \mathcal{A}$ iff $\mathcal{I}^{\mathcal{U}} \models \mathcal{A}^{\mathcal{U}}$.*

Similar to the concepts $C^{\mathcal{U}}$, the ABox update $\mathcal{A}^{\mathcal{U}}$ works only if the interpretations $\mathcal{I}$ of $\mathcal{A}$ satisfy no assertion in $\mathcal{U}$. For a fixed interpretation $\mathcal{I}$, we can overcome this problem by replacing $C^{\mathcal{U}}$ with $C^{\mathcal{U}'}$, where $\mathcal{U}'$ is the set of those assertions in $\mathcal{U}$ that are violated in $\mathcal{I}$. However, in general we are confronted with the problem that an ABox can have many different models, and these models can violate different assertions of the update $\mathcal{U}$. Hence, there is no unique way of moving from $C^{\mathcal{U}}$ to $C^{\mathcal{U}'}$ as described above.

$$
\begin{array}{rcl}
(\exists r.C)^{\mathcal{U}} & = & (\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \sqcap \exists r.C^{\mathcal{U}}) \sqcup \exists r.(\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \sqcap C^{\mathcal{U}}) \\
& & \sqcup \bigsqcup_{a,b\in \mathsf{Obj}(\mathcal{U}), r(a,b)\notin\mathcal{U}} (\{a\} \sqcap \exists r.(\{b\} \sqcap C^{\mathcal{U}})) \sqcup \bigsqcup_{\neg r(a,b)\in\mathcal{U}} (\{a\} \sqcap @_b C^{\mathcal{U}}) \\
(\forall r.C)^{\mathcal{U}} & = & (\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \rightarrow \forall r.C^{\mathcal{U}}) \sqcap \forall r.(\bigsqcap_{a\in \mathsf{Obj}(\mathcal{U})} \neg\{a\} \rightarrow C^{\mathcal{U}}) \\
& & \sqcap \bigsqcap_{a,b\in \mathsf{Obj}(\mathcal{U}), r(a,b)\notin\mathcal{U}} (\{a\} \rightarrow \forall r.(\{b\} \rightarrow C^{\mathcal{U}})) \sqcap \bigsqcap_{\neg r(a,b)\in\mathcal{U}} (\{a\} \rightarrow @_b C^{\mathcal{U}})
\end{array}
$$

Figure 16: Constructing $C^{\mathcal{U}}$ for existential and universal restrictions

The solution is to produce an updated ABox for each subset $\mathcal{U}' \subseteq \mathcal{U}$ of violated assertions separately, and then simply take the disjunction.

Let $\mathcal{A}$ be an ABox and $\mathcal{U}$ an update. A simple ABox $\mathcal{D}$ is called a *diagram for* $\mathcal{U}$ if it is a maximal consistent subset of $L_{\mathcal{U}}$, where $L_{\mathcal{U}} = \{\psi, \neg\psi \mid \psi \in \mathcal{U}\}$ is the set of *literals* over $\mathcal{U}$. Intuitively, a diagram gives a complete description of the part of a model of $\mathcal{A}$ that is "relevant" for the update $\mathcal{U}$. Let $\mathfrak{D}$ be the set of all diagrams for $\mathcal{U}$ and consider for $\mathcal{D} \in \mathfrak{D}$ the set $\mathcal{D}_{\mathcal{U}} := \{\psi \mid \neg\psi \in \mathcal{D} \text{ and } \psi \in \mathcal{U}\}$ which corresponds to taking a subset of $\mathcal{U}$ as described above: we retain only those parts of $\mathcal{U}$ that are violated by interpretations whose relevant part is described by $\mathcal{D}$. We now define the updated ABox $\mathcal{A}'$ as

$$\mathcal{A}' = \bigvee_{\mathcal{D}\in\mathfrak{D}} \bigwedge \mathcal{A}^{\mathcal{D}_{\mathcal{U}}} \cup \mathcal{D}_{\mathcal{U}} \cup (\mathcal{D} \setminus \neg\mathcal{D}_{\mathcal{U}}).$$

Here, we use Boolean ABox operators only as an abbreviation for the "@" constructor. The expansion of this abbreviation does not substantially increase the size of the updated ABox: the translation from Boolean ABoxes to non-Boolean ones described in [101] is linear. To achieve a less redundant ABox, it is possible to drop from $\mathcal{A}'$ those disjuncts for which the diagram $\mathcal{D}$ is not consistent w.r.t. $\mathcal{A}$. This is, however, not strictly necessary since the ABox $\mathcal{D} \setminus \neg\mathcal{D}_{\mathcal{U}}$ ensures that these disjuncts are inconsistent.

Lemma 89. $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$.

It is easy to adapt the construction of updated ABoxes to the DLs $\mathcal{ALCO}^{@}$, $\mathcal{ALCIO}^{@}$, $\mathcal{ALCQO}^{@}$. For the former two, we have to treat existential and universal restrictions in the $C^{\mathcal{U}}$ translation rather than number restrictions. The corresponding clauses are shown in Figure 16. The lemmas proved above for $\mathcal{ALCQIO}^{@}$ are then easily adapted.

Theorem 90. *All of the following DLs have ABox updates:* $\mathcal{ALCO}^{@}$, $\mathcal{ALCIO}^{@}$, $\mathcal{ALCQO}^{@}$, *and* $\mathcal{ALCQIO}^{@}$.

Now that we know that updated ABoxes always exist in the above DLs, we should have a look at their size. Let us first make precise what we mean with this. The *length* of a concept C, denoted by $|C|$, is the number of symbols needed to write C. Note that it makes a considerable difference whether we assume the numbers inside number restrictions to be written in unary or in binary: if written in unary, we have $|(\leqslant n\ r\ C)| \in \mathcal{O}(n)$, and if written in binary, we have $|(\leqslant n\ r\ C)| \in \mathcal{O}(\log n)$. In the following, we will always point out to which coding our results apply.[27] The *size* of an ABox assertion $C(a)$ is $|C|$, the size of $r(a,b)$ and $\neg r(a,b)$ is 1. Finally, the *size* of an ABox $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the sum of the sizes of all assertions in $\mathcal{A}$.

[27] In fact, all results except Theorems 99 and 100 apply to both unary and binary coding.

A close inspection of our construction reveals the following: first, the size the concepts $C^{\mathcal{D}_\mathcal{U}}$ is exponential in the size of $\mathcal{A}$ and polynomial in the size of $\mathcal{U}$; and second, the number of disjuncts in $\mathcal{A}'$ is exponential in the size of $\mathcal{U}$. These bounds hold regardless of the coding of numbers.

Theorem 91.
Let $\mathcal{L} \in \{\mathcal{ALCO}^{@}, \mathcal{ALCIO}^{@}, \mathcal{ALCQO}^{@}, \mathcal{ALCQIO}^{@}\}$. Then there are polynomials p_1, p_2, and q such that, for every $\mathcal{L}$ ABox $\mathcal{A}$ and every update $\mathcal{U}$, there is an $\mathcal{L}$ ABox $\mathcal{A}'$ such that

- $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$;
- $|\mathcal{A}'| \leq 2^{p_1(|\mathcal{A}|)} \cdot 2^{p_2(|\mathcal{U}|)}$;
- $\mathcal{A}'$ *can be computed in time* $q(|\mathcal{A}'|)$.

There are applications in which the domain of interest evolves continuously. In such an environment, it is necessary to update an ABox over and over again. Then, it is clearly important that the exponential blowups of the individual updates do not add up. The following theorem, which can be proved by carefully investigating our update construction, shows that this is indeed not the case. It holds independently of the coding of numbers.

Theorem 92. *There are polynomials p_1, p_2 such that the following holds: for all ABoxes $\mathcal{A}_0, \ldots, \mathcal{A}_n$ and updates $\mathcal{U}_1, \ldots, \mathcal{U}_n$, if $\mathcal{A}_i$ is the ABox computed by our algorithm when $\mathcal{A}_{i-1}$ is updated with $\mathcal{U}_i$, for $0 < i \leq n$, then*

$$|\mathcal{A}_n| \leq 2^{p_1(|\mathcal{A}_0|)} \cdot 2^{p_2(|\mathcal{U}_1|+\cdots+|\mathcal{U}_n|)}.$$

8.5.1 Conditional Updates

For the sake of simplicity, we have defined ABox updates to be unconditional: the assertions in the update $\mathcal{U}$ are unconditionally true after the update and we cannot express statements such as "$A(a)$ is true after the update if $C(b)$ was true before". In some applications such as reasoning about actions with DLs [10], it is more useful to have *conditional updates*, where the initial interpretation determines the changes that are triggered.

A *conditional update* $\mathcal{U}$ is a finite set of expressions φ/ψ, where the *precondition* φ is an ABox assertion (possibly involving non-atomic concepts) and the *postcondition* ψ is an assertion of the form

$$A(a), \neg A(a), r(a,b), \neg r(a,b)$$

with A a concept name. Intuitively, an expression φ/ψ means that if φ holds in the initial interpretation, then ψ holds after the update. As in the case of unconditional updates, we require a consistency condition: if φ/ψ and $\varphi'/\neg\psi$ are both in $\mathcal{U}$, then the ABox $\{\varphi, \varphi'\}$ has to be inconsistent.

The definition of "$\Longrightarrow_\mathcal{U}$" is easily adapted to the case of conditional updates: for $\mathcal{U}$ a conditional update, we write $\mathcal{I} \Longrightarrow_\mathcal{U} \mathcal{I}'$ if the following hold:

- for all concept names A,
  $A^{\mathcal{I}'} = (A^\mathcal{I} \cup \{a^\mathcal{I} \mid \varphi/A(a) \in \mathcal{U} \wedge \mathcal{I} \models \varphi\}) \setminus \{a^\mathcal{I} \mid \varphi/\neg A(a) \in \mathcal{U} \wedge \mathcal{I} \models \varphi\}$
- for all role names r,
  $r^{\mathcal{I}'} = (r^\mathcal{I} \cup \{(a^\mathcal{I}, b^\mathcal{I}) \mid \varphi/r(a,b) \in \mathcal{U} \wedge \mathcal{I} \models \varphi\}) \setminus \{(a^\mathcal{I}, b^\mathcal{I}) \mid \varphi/\neg r(a,b) \in \mathcal{U} \wedge \mathcal{I} \models \varphi\}$.

Then the result of updating an ABox is defined exactly as in the case of unconditional updates. Clearly, conditional updates generalize unconditional once since assertions ψ of unconditional updates can be expressed as $\top(a)/\psi$, with a an arbitrary individual name.

We now show how to adapt our construction of updated ABoxes to conditional updates. For $\mathcal{U}$ a conditional update, we use $\mathsf{rhs}(\mathcal{U})$ to denote $\{\psi \mid \varphi/\psi \in \mathcal{U}\}$, and $\mathsf{lhs}(\mathcal{U})$ for $\{\varphi \mid \varphi/\psi \in \mathcal{U}\}$. In the original algorithm, the updated ABox $\mathcal{A}'$ is assembled by taking one disjunct for every diagram for $\mathcal{U}$. The intuition is that, when a diagram $\mathcal{D}$ is satisfied by an interpretation $\mathcal{I}$, then we know which assertions in $\mathcal{U}$ have already been satisfied in $\mathcal{I}$ before $\mathcal{U}$ is applied. We generalize this idea to conditional updates by taking one disjunct for each pair $(\mathcal{D},\mathcal{U}')$, where $\mathcal{D}$ is a diagram for $\mathsf{rhs}(\mathcal{U})$, and $\mathcal{U}'$ is a subset of $\mathcal{U}$. Intuitively, $\mathcal{U}'$ determines the set of assertions from $\mathcal{U}$ whose preconditions are satisfied in the initial model, and $\mathcal{D}$ determines the post-conditions that actually cause a change.

Let $\mathfrak{D}$ be the set of all diagrams for $\mathsf{rhs}(\mathcal{U})$. Let $\mathcal{D} \in \mathfrak{D}$ and $\mathcal{U}' \subseteq \mathcal{U}$. We define

$$\mathcal{D}_{\mathcal{U}'} := \{\psi \mid \neg\psi \in \mathcal{D} \text{ and } \varphi/\psi \in \mathcal{U}'\}.$$

Then we can assemble the updated ABox $\mathcal{A}'$ as follows:

$$\begin{aligned}\mathcal{A}' \;=\; \bigvee_{\mathcal{D}\in\mathfrak{D}} \bigvee_{\mathcal{U}'\subseteq\mathcal{U}} \bigwedge & \{\varphi \mid \varphi/\psi \in \mathcal{U}'\}^{\mathcal{D}_{\mathcal{U}'}} \\ & \cup \{\neg\varphi \mid \varphi/\psi \in \mathcal{U} \setminus \mathcal{U}'\}^{\mathcal{D}_{\mathcal{U}'}} \\ & \cup \mathcal{A}^{\mathcal{D}_{\mathcal{U}'}} \cup \mathcal{D}_{\mathcal{U}'} \cup (\mathcal{D} \setminus \neg\mathcal{D}_{\mathcal{U}'}).\end{aligned}$$

The notion of a description logic $\mathcal{L}$ *having conditional ABox updates* is defined in the obvious way.

Theorem 93. *All of the following DLs have conditional ABox updates: $\mathcal{ALCO}^{@}$, $\mathcal{ALCIO}^{@}$, $\mathcal{ALCQO}^{@}$, and $\mathcal{ALCQIO}^{@}$.*

Concerning the size and computability of updated ABoxes, we obtain the same bounds as in Theorem 91, independently of the coding of numbers.

8.6 A Lower Bound for the Size of Updated ABoxes

In the following sections, we are interested in the question whether or not the exponential blowup observed in Theorems 91 and 92 can be avoided. In this section, we consider updates of *propositional logic theories* where the updates are of the restricted form considered in this section, i.e., conjunctions of literals. We prove that, even in this case, an exponential blowup in the size of the whole input (original ABox + update) cannot be avoided unless the complexity classes PTIME and NC coincide. As discussed in [122], this is believed to be similarly unlikely as PTIME $=$ NP. It is not difficult to prove that this lower bound on the size of updated ABoxes transfers to all DLs considered in this section.

For the following definitions, we fix an individual name a. A *propositional ABox* $\mathcal{A}$ is of the form $\{C(a)\}$ with C a *propositional concept*, i.e., a concept that uses only the concept constructors $\neg$, $\sqcap$, and $\sqcup$. A *propositional update* $\mathcal{U}$ contains only assertions of the form $A(a)$ and $\neg A(a)$. Observe that propositional ABoxes and propositional updates are only allowed to refer to the single, fixed individual name a.

For the semantics, we fix a single individual x. Since we are dealing with propositional ABoxes and updates, we assume that interpretations do not intepret role names, and that interpretation domains have only a single element x with $a^{\mathcal{I}} = x$. We introduce a couple of notions. For a concept C, let $\mathsf{C}(C)$ denote the set of concept names used in C. For an interpretation $\mathcal{I}$ and a set of concept names Γ, let $\mathcal{I}|_\Gamma$ denote the restriction of $\mathcal{I}$ that interpretes only the concept names in Γ. Let C be a concept and $\Gamma \subseteq \mathsf{C}(C)$. Then

a concept D is called a *uniform* Γ*-interpolant* of C iff $\mathsf{C}(D) \subseteq \Gamma$ and $\{\mathcal{I}|_\Gamma \mid x \in C^\mathcal{I}\} = \{\mathcal{I}|_\Gamma \mid x \in D^\mathcal{I}\}$. It is easily seen that, for any propositional concept C and subset $\Gamma \subseteq \mathsf{C}(C)$, the uniform Γ-interpolant of C exists and is unique up to equivalence. The following lemma establishes a tight connection between uniform interpolants and propositional updates.

Lemma 94. *Let $\mathcal{A} = \{C(a)\}$ be a propositional ABox, $\mathcal{U}$ a propositional update, Γ the set of concept names in C not occurring in $\mathcal{U}$, $\widehat{C}$ the shortest uniform Γ-interpolant of C, and*

$$\mathcal{A}' = \{a : (\widehat{C} \sqcap \bigsqcap_{A(a)\in\mathcal{U}} A)\}.$$

Then we have the following:
(i) $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$;
(ii) if $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}''$, then $|\mathcal{A}'| \leq |\mathcal{U}| + |\mathcal{A}''|$.

It thus remains to show that the size of (smallest) uniform interpolants of propositional concepts is not bounded polynomially in the size of the interpolated concept unless PTIME = NC.

The size of uniform interpolants of propositional concepts is closely related to the relative succinctness of propositional logic (PL) formulas and Boolean circuits. We remind that both PL formulas and Boolean circuits compute Boolean functions and refer, e.g., to [122] for exact definitions. We use $|c|$ to denote the number of gates in the Boolean circuit c, and $|\varphi|$ to denote the length of the PL formula φ. It is known that, unless PTIME = NC, there exists no polynomial p such that every Boolean circuit c can be converted into a PL formula φ that computes the same function as c_i and satisfies $|\varphi| \leq p(|c_i|)$, see e.g. Exercise 15.5.4 of [122]. In the following, we show that non-existence of such a polynomial p implies that uniform interpolants are not bounded polynomially in the size of the interpolated concept. Take a Boolean circuit c with k inputs. Then c can be translated into a propositional concept D_c by introducing concept names $I_1, \ldots, I_k$ for the inputs and, additionally, one auxiliary concept name for the output of every gate. Let $\mathcal{G}$ be the set of concept names introduced for gate outputs, and let $O \in \mathcal{G}$ be the concept name for the output of the gate computing the final output of c. It is not difficult to see that this translation can be done such that there exists a polynomial q such that, for all Boolean circuits c,

(i) $|D_c| \leq q(|c|)$ and

(ii) for all interpretations $\mathcal{I}$ and all $x \in D_c^\mathcal{I}$, $x \in O^\mathcal{I}$ iff c outputs "true" on input $b_1, \ldots, b_k$, where $b_j = 1$ if $x \in I_j^\mathcal{I}$ and $b_j = 0$ otherwise.

Now, set $\Gamma := \mathcal{G} \setminus \{O\}$. Then the uniform Γ-interpolant $\widehat{D}_c$ of D_c also satisfies (ii). Thus, $\widehat{D}_c$ is a (notational variant of a) propositional logic formula computing the same Boolean function as c. If the size of $\widehat{D}_c$ would be bounded polynomially in the size of D_c, we thus had obtained a contradiction to our assumption on the non-existence of the polynomial p. Together with Lemma 94, we obtain the following theorem.

Theorem 95. *Unless* PTIME = NC*, there exists no polynomial p such that, for all propositional ABoxes $\mathcal{A}$ and propositional updates $\mathcal{U}$, there exists a propositional ABox $\mathcal{A}'$ such that*

- *$\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$ and*
- *$|\mathcal{A}'| \leq p(|\mathcal{A}| + |\mathcal{U}|)$.*

Our result is closely related to a result of [27] who prove an analogue of Theorem 95 under the complexity-theoretic assumption that the polynomial hierarchy does not collapse. However, Cadoli et al.'s technique does not appear to work with the restricted form of updates (conjunctions of literals) considered in this section.

8.7 Small(er) Updated ABoxes

Theorem 91 does not differentiate between exponential blowups in the size of the original ABox and exponential blowups in the size of the update. In contrast to the former, the latter is usually considered acceptable since updates will usually be small compared to the original ABox $\mathcal{A}$. We believe that, in the DLs mentioned in Theorem 90, the exponential blowup in the size of $\mathcal{A}$ is unavoidable. However, we have to leave a proof as an open problem. In the following, we exhibit three different ways to extend $\mathcal{ALCQIO}^{@}$ and its fragments such that it becomes possible to compute updated ABoxes that are only polynomial in the size of the original ABox.

A first, rather restrictive solution is to admit only concept assertions in updates. Then, in all DLs captured by Theorem 90, computing the concepts $C^{\mathcal{U}}$ becomes a lot simpler: just replace every concept name A in C with

$$A \sqcup \bigsqcup_{\neg A(a) \in \mathcal{B}} \{a\} \sqcap \neg (\bigsqcup_{A(a) \in \mathcal{B}} \{a\}).$$

If modified in this way, our original construction yields updated ABoxes that are only polynomial in the size of the original ABox (but still exponential in $\mathcal{U}$). The bound is independent of the coding of numbers and also applies to iterated updates.

8.7.1 Small Updates Through TBoxes

We show how to produce smaller updated ABoxes by allowing the introduction of auxiliary concept names via an acyclic TBox. In the propositional case, this corresponds to admitting additional variables for defining abbreviations. In the terminology of Cadoli et al. [27], we thus move from logical equivalence to query equivalence. In this way, we obtain updates that are polynomial in the size of the original ABox.

In the following, we assume that the set of concept names is partitioned into a set of *primary* concept names and a set of *auxiliary* concept names. The latter are used only for defining abbreviations in a TBox. A *concept definition* is of the form $A \equiv C$, where A is an auxiliary concept name and C is a concept. An *(acyclic) TBox* $\mathcal{T}$ is a finite set of concept definitions with unique left-hand sides and without cyclic definitions [8], page 52. We call a concept name A *defined* in a TBox $\mathcal{T}$ and write $A \in \mathsf{def}(\mathcal{T})$ if A occurs on the left-hand side of a concept definition in $\mathcal{T}$. A *knowledge base (KB)* is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ such that every auxiliary concept name used in $\mathcal{K}$ is in $\mathsf{def}(\mathcal{T})$. An interpretation $\mathcal{I}$ *satisfies* a concept definition $A \equiv C$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$, written $\mathcal{I} \models \mathcal{T}$, if $\mathcal{I}$ satisfies all concept definitions in $\mathcal{T}$. An interpretation $\mathcal{I}$ is a *model* of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{A}$. The set of all models of a KB $\mathcal{K}$ is denoted $M(\mathcal{K})$.

An *update* $\mathcal{U}$ is a simple and consistent ABox that does not use auxiliary concept names. We disallow auxiliary concept names because they can be defined in a TBox and thus allowing them is equivalent to admitting updates with complex concepts. Let $\mathcal{U}$ be an update and let $\mathcal{I}$ and $\mathcal{I}'$ be interpretations that agree on the interpretation of individual names. We define an update relation $\mathcal{I} \Longrightarrow^{p}_{\mathcal{U}} \mathcal{I}'$ (where p stands for "primary") as in Definition 74, but restrict the condition on concept names to primary concept names. This restriction is not harmful since we require auxiliary concept names that are used in a knowledge base to be defined in the TBox, and this means that their extension is uniquely determined by the extensions

of the primary concept names and role names. Still, as a result of the restriction, the relation $\Longrightarrow^p_{\mathcal{U}}$ is not functional (in contrast to the case without TBoxes).

Definition 96 (Knowledge Base Update)**.** Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be knowledge bases, $\mathcal{K}_i = (\mathcal{T}_i, \mathcal{A}_i)$, and $\mathcal{U}$ an update. Then $\mathcal{K}_2$ *is a result of updating* $\mathcal{K}_1$ *with* $\mathcal{U}$ if

$$M(\mathcal{K}_2) = \{\mathcal{I}' \mid \exists \mathcal{I} \in M(\mathcal{K}_1) : \mathcal{I} \Longrightarrow^p_{\mathcal{U}} \mathcal{I}' \wedge \mathcal{I}' \models \mathcal{T}_2\}.$$

In this case, we write $\mathcal{K}_1 * \mathcal{U} \equiv_p \mathcal{K}_2$. △

Observe that the TBox of the updated KB $\mathcal{K}_2$ can contain new abbreviations, i.e., definitions $A \doteq C$ with A an auxiliary concept names that does not occur in $\mathcal{K}_1$. Since there is more than a single way to define such abbreviations, the result of updating a knowledge base is not unique up to logical equivalence. However, we have this uniqueness when restricting our attention to what the updated ABox expresses regarding the primary concept names and role names, only.

In the equality in Definition 96, the conjunct $\mathcal{I}' \models \mathcal{T}_2$ has no impact on the "$\subseteq$" direction since all models of $\mathcal{K}_2$ are models of $\mathcal{T}_2$ anyway. For the "$\supseteq$" direction, the conjunct is essential: dropping it would mean to require that *every* possible interpretation of the auxiliary concept names in $\mathcal{I}'$ satisfies $\mathcal{T}_2$. Moreover, since $\mathcal{T}_2$ is part of the updated knowledge base $\mathcal{K}_2$, interpretations not satisfying $\mathcal{T}_2$ are irrelevant.

We now establish a relationship between updates of ABoxes and updates of knowledge bases. Let $\mathcal{T}$ be an acyclic TBox, and C a concept. The concept $C^{\mathcal{T}}$ obtained from C by exhaustively replacing defined concept names in C with their definitions from $\mathcal{T}$ is called the *unfolding of* C *w.r.t.* $\mathcal{T}$. If $\mathcal{A}$ is an ABox, then the *unfolding of* $\mathcal{A}$ *w.r.t.* $\mathcal{T}$ is the ABox $\mathcal{A}^{\mathcal{T}}$ obtained by replacing each concept assertion $C(a)$ in $\mathcal{A}$ with $C^{\mathcal{T}}(a)$. If $(\mathcal{T}, \mathcal{A})$ is a knowledge base, then the unfolding $\mathcal{A}^{\mathcal{T}}$ contains only primary concept names. The following lemma shows that updated knowledge bases are just updated ABoxes with abbreviations.

Lemma 97. *Let* $\mathcal{K}_1$ *and* $\mathcal{K}_2$ *be knowledge bases,* $\mathcal{K}_i = (\mathcal{T}_i, \mathcal{A}_i)$*, and* $\mathcal{U}$ *an update. Then*

$$\mathcal{K}_1 * \mathcal{U} \equiv_p \mathcal{K}_2 \quad \textit{iff} \quad \mathcal{A}_1^{\mathcal{T}_1} * \mathcal{U} \equiv \mathcal{A}_2^{\mathcal{T}_2}.$$

For the moment, the purpose of Lemma 97 is only to clarify the relation between ABox updates and knowledge base updates. Although we could compute knowledge base updates using Lemma 97 together with our construction for ABox updates, this would not help to obtain smaller updates.

Therefore, we now show how to directly construct updated knowledge bases in $\mathcal{ALCQIO}^{@}$ and its fragments. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, and let $\mathcal{U}$ be an update. Diagrams for $\mathcal{U}$ and the sets $\mathfrak{D}$ and $\mathcal{D}_{\mathcal{U}}$ are defined as in the previous section. We use $\mathsf{sub}(\mathcal{K})$ to denote the set of all subconcepts of concepts occurring in $\mathcal{K}$. To construct the result of updating $\mathcal{K}$ with $\mathcal{U}$, we introduce a new concept name $A_C^{\mathcal{D}}$ for every diagram $\mathcal{D} \in \mathfrak{D}$ and every $C \in \mathsf{sub}(\mathcal{K})$. Let $\mathsf{trans}(C, \mathcal{D})$ denote the concept on the right-hand side of the clause for $C^{\mathcal{D}_{\mathcal{U}}}$ in Figure 15, with all subconcepts $E^{\mathcal{D}}$ replaced by the concept name $A_E^{\mathcal{D}}$. For example, $\mathsf{trans}(C \sqcap D, \mathcal{D}) = A_C^{\mathcal{D}} \sqcap A_D^{\mathcal{D}}$. For each diagram $\mathcal{D} \in \mathfrak{D}$, define a TBox

$$\mathcal{T}_{\mathsf{sub}}^{\mathcal{D}} := \{A_C^{\mathcal{D}} \equiv \mathsf{trans}(C, \mathcal{D}) \mid C \in \mathsf{sub}(\mathcal{K}) \setminus \mathsf{def}(\mathcal{T})\}.$$

Then, we define the TBox

$$\mathcal{T}' := \bigcup_{\mathcal{D} \in \mathfrak{D}} (\mathcal{T}_{\mathsf{sub}}^{\mathcal{D}} \cup \{A_A^{\mathcal{D}} \equiv A_C^{\mathcal{D}} \mid A \equiv C \in \mathcal{T}\}).$$

For every $\mathcal{D} \in \mathfrak{D}$, let

$$\begin{aligned}\mathcal{A}_{\mathcal{D}_\mathcal{U}} \quad := \quad & \{A_C^\mathcal{D}(a) \mid C(a) \in \mathcal{A}\} \cup \cup \\ & \{r(a,b) \mid r(a,b) \in \mathcal{A} \wedge \neg r(a,b) \notin \mathcal{D}_\mathcal{U}\} \cup \\ & \{\neg r(a,b) \mid \neg r(a,b) \in \mathcal{A} \wedge r(a,b) \notin \mathcal{D}_\mathcal{U}\}\end{aligned}$$

Now we can define the ABox $\mathcal{A}'$ by setting

$$\mathcal{A}' = \bigvee_{\mathcal{D} \in \mathfrak{D}} \bigwedge \mathcal{A}_{\mathcal{D}_\mathcal{U}} \cup \mathcal{D}_\mathcal{U} \cup (\mathcal{D} \setminus \neg \mathcal{D}_\mathcal{U}).$$

and finally assemble the updated knowledge base by setting $\mathcal{K}' := (\mathcal{T}', \mathcal{A}')$. It can be proved that this knowledge base is as required:

Lemma 98. $\mathcal{K} * \mathcal{U} \equiv_p \mathcal{K}'$

We now formulate the main result on updates with acyclic TBoxes. In constrast to updates without TBoxes, updated knowledge bases are polynomial in the size of the original KB. Thus, Lemma 97 implies that we can use acyclic TBoxes to obtain a more succinct presentation of updated ABoxes. In the following, the size $|\mathcal{T}|$ of a TBox $\mathcal{T}$ is $\sum_{A \equiv C \in \mathcal{T}} |C|$, and the size $|\mathcal{K}|$ of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is the sum of $|\mathcal{T}|$ and $|\mathcal{A}|$.

Theorem 99.
Let $\mathcal{L} \in \{\mathcal{ALCO}^@, \mathcal{ALCIO}^@, \mathcal{ALCQO}^@, \mathcal{ALCQIO}^@\}$. Then there are polynomials p_1, p_2, and q such that, for every $\mathcal{L}$-knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and every update $\mathcal{U}$, there is an $\mathcal{L}$-knowledge base $\mathcal{K}'$ such that

- $\mathcal{K} * \mathcal{U} \equiv_p \mathcal{K}'$;
- $|\mathcal{K}'| \leq p_1(|\mathcal{K}|) \cdot 2^{p_2(|\mathcal{U}|)}$;
- *$\mathcal{K}'$ can be computed in time $q(|\mathcal{K}'|)$.*

It is important to note that Theorem 99 is true only if we assume unary coding of numbers: with binary coding, already the translation C^U results in an exponential blowup in the size of the original ABox since we have $|(\leqslant n\ r\ C)^\mathcal{U}| \in \mathcal{O}(2^n)$. Thus, the updated ABox will not be polynomial in the size of the original one.

As in the case without TBoxes, it can be shown that iterated updates do not produce a blowup of the size of updated ABoxes that is worse than the blowup produced by a single update.

Theorem 100. *There are polynomials p_1, p_2 such that the following holds: for all knowledge bases $\mathcal{K}_0, \ldots, \mathcal{K}_n$ and updates $\mathcal{U}_1, \ldots, \mathcal{U}_n$, if $\mathcal{K}_i$ is the ABox computed by our algorithm when $\mathcal{K}_{i-1}$ is updated with $\mathcal{U}_i$, for $0 < i \leq n$, then*

$$|\mathcal{K}_n| \leq p_1(|\mathcal{K}_0|) \cdot 2^{p_2(|\mathcal{U}_1| + \cdots + |\mathcal{U}_n|)}.$$

8.7.2 Small Updates in $\mathcal{ALCQIO}^+$

We have argued above that, if the update contains no role assertions, then updated $\mathcal{ALCQIO}^@$ ABoxes are polynomial in the size of the original ABox even without introducing TBoxes. Intuitively, updates with only concept assertions do not lead to an exponential blowup because we have available nominals, the @-operator, and the Boolean operators on concepts. In standard DLs, none of these operators is available for roles: we can neither construct the union of roles, nor their complement, nor a "nominal role" $\{(a, b)\}$ with a and b nominals. In this section, we explore the possibility of constructing updated ABoxes in a language in which such constructors are available. The language we consider is of almost the same expressive power as C^2, the two-variable fragment of first-order logic with counting quantifiers [103].

Denote by $\mathcal{ALCQIO}^+$ the description logic extending $\mathcal{ALCQIO}^@$ by means of the role constructors $\cap$ (role intersection), $-$ (set-theoretic difference of roles), and $\{(a, b)\}$ (nominal roles). In this language, complex roles are constructed starting from role names and nominal roles, and then applying $\cap$, $-$, and the inverse role operator $\cdot^-$. The interpretation of complex roles is as expected:

- $\{(a, b)\}^{\mathcal{I}} = \{(a^{\mathcal{I}}, b^{\mathcal{I}})\}$, for all $a, b \in \mathsf{N_I}$;
- $(r_1 \cap r_2)^{\mathcal{I}} = r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}}$;
- $(r_1 - r_2)^{\mathcal{I}} = r_1^{\mathcal{I}} - r_2^{\mathcal{I}}$.

We note that reasoning in $\mathcal{ALCQIO}^+$ is decidable: this DL can easily be embedded into C^2 and, therefore, ABox consistency is decidable in NEXPTIME even if the numbers inside number restrictions are coded in binary [121, 125]. We now formulate the main result of this section:

Theorem 101. *There are polynomials p_1, p_2, and q such that, for every $\mathcal{ALCQIO}^+$ ABox $\mathcal{A}$ and every update $\mathcal{U}$, there is an $\mathcal{ALCQIO}^+$ ABox $\mathcal{A}'$ such that*

- $\mathcal{A} * \mathcal{U} \equiv \mathcal{A}'$;
- $|\mathcal{A}'| \leq p_1(|\mathcal{A}|) \cdot 2^{p_2(|\mathcal{U}|)}$;
- $\mathcal{A}'$ *can be computed in time* $q(|\mathcal{A}'|)$.

Proof. We modify the proof of Theorem 91. For $\mathcal{ALCQIO}^+$, the construction of the concepts $C^{\mathcal{U}}$ is much simpler: it suffices to replace every concept name A in C with

$$A \sqcup \bigsqcup_{\neg A(a) \in \mathcal{U}} \{a\} \sqcap \neg(\bigsqcup_{A(a) \in \mathcal{U}} \{a\})$$

and every role name r in C with

$$r \cup \bigcup_{\neg r(a,b) \in \mathcal{U}} \{(a, b)\} \setminus (\bigcup_{r(a,b) \in \mathcal{U}} \{(a, b)\}).$$

The concepts $C^{\mathcal{U}}$ are of size polynomial in the size of C and $\mathcal{U}$. The ABox $\mathcal{A}'$ can then be constructed in the same way as in the proof of Theorem 91 and is polynomial in the size of $\mathcal{A}$, but exponential in the size of the update $\mathcal{U}$. ❑

Clearly, Theorem 101 is independent of the coding of numbers, and, also with iterated updates, updated ABoxes remain polynomial in the size of the original ABox. An alternative to working with a description logic such as $\mathcal{ALCQIO}^+$ is to work directly in the two-variable fragment with counting C^2. Then, a result analogous to Theorem 101 is easily obtained.

8.8 Outlook

There are two obvious directions for future work. The first direction is to alleviate the syntactic restriction posed on concepts appearing in updates in a controlled way. For example, research on propositional updates containing *disjunctions* [161, 71, 99, 141] suggests the feasibility of ABox updates with Boolean combinations of concept names. We conjecture that natural generalizations of the semantics proposed in the propositional case lead to useful notions of an ABox update under which the updates are still computable. The second direction for future work is to incorporate cyclic TBoxes into our framework. However, this direction appears to be considerably more difficult than the first one. As discussed in [10], it is not even clear if a satisfactory semantics can be defined in this case.

9 Semantic Service Discovery and Selection

9.1 Introduction

Description logics play an important rôle in the Semantic Web since they are the basis of the W3C-recommended Web ontology language OWL [16, 83], which can be used to create semantic annotations describing the content of Web pages [139].

In addition to this static information, the Web also offers services, which allow their users to effect changes in the world, such as 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.

The OWL-S initiative [138] uses OWL to develop an ontology of services, covering different aspects of Web services, among them functionality. To describe their functionality, services are viewed as processes that (among other things) have pre-conditions and effects. However, the faithful representation of the dynamic behaviour of such processes (what changes of the world they cause) is beyond the scope of a static ontology language like OWL.

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. [128, 140]). Thus, it is not surprising that theories developed in these comunities have been applied in the context of Semantic Web services. For example, [108, 109] use the situation calculus [128] and GOLOG [96] to formalize the dynamic aspects of Web services and to describe their composition. In [136], OWL-S process models are translated into the planning language of the HTN planning system SHOP2 [115], which is then used for automatic Web service composition.

The approach used in this paper is in a similar vein. We are interested in the faithful description of the changes to the world induced by the invocation of a service. To this purpose, we describe services as actions that have pre-conditions and post-conditions (its effects). These conditions are expressed with the help of description logic assertions, and the current state of the world is (incompletely) described using a set of such assertions (a so-called 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 [159, 160, 158, 48, 71], and is fully compatible with the usual DL semantics. However, we will also show that this semantics can be viewed as an instance of Reiter's approach [127, 123, 95, 128] for taming the situation calculus. In particular, our semantics solves the frame problem in precisely the same way.

Then, 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 contribution of this paper is an analysis of how the choice of the DL influences the complexity of these two reasoning tasks for services. 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). The reason is that we can reduce both tasks for services to the standard DL task of

checking consistency of an ABox w.r.t. an acyclic TBox, provided that we can use nominals within concept descriptions. This reduction is optimal since our hardness results show that the complexity increase (sometimes) caused by the addition of nominal cannot be avoided. We also motivate the restrictions we impose: we discuss the semantic and the computational problems that arise when these restrictions are loosened. Most importantly, we prove that allowing for complex concepts in post-conditions not only yields semantic problems, but also the undecidability of the two service reasoning problems.

The results from this section are publishad as a part of [10] Because of the space constraints, all proofs and a more detailed discussion of the relationship to the situation calculus must be omitted. They can be found in [11].

9.2 Service Descriptions

The framework for reasoning about Web services proposed in this section is not restricted to a particular description logic, but can be instantiated with any description logic that seems appropriate for the application domain at hand. For our complexity results, we consider the DL $\mathcal{ALCQIO}$ and a number of its sublanguages. The reason for choosing $\mathcal{ALCQIO}$ is that it forms the core of OWL-DL, the description logic variant of OWL. The additional OWL-DL constructors could be easily added, with the exception of transitive roles which are discussed in Section 9.5.

We now introduce the formalism for reasoning about Web services. For simplicity, we concentrate 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. The handling of such parametric services takes place "outside" of our formalism and is not discussed in detail in the current paper. We may restrict ourselves to ground services since all the reasoning tasks considered in this paper presuppose that parametric services have already been instantiated. For other tasks, such as planning, it may be more natural to work directly with parametric services.

Definition 102 (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. If φ is tautological, e.g. $\top(a)$ for some individual name a, then we write just ψ instead of φ/ψ. By the law of inertia, only those facts that are forced to change by the post-conditions should be changed by applying the service. However, it is well-known in the reasoning about action community that enforcing this minimization of change strictly is sometimes too restrictive [98, 132]. The rôle of occlusions is to describe those primitive literals to which the minimization condition does not apply.

To illustrate the definition of services, consider a Web site offering services for people who move from Continental Europe to the United Kingdom. Among its services are getting a contract with an

electricity provider, opening a bank account, and applying for child benefit. Obtaining an electricity contract b for customer a does not involve any pre-conditions. It is described by the service S_1, which has an empty set of pre-conditions, an empty set of occlusions, and whose post-conditions are defined as follows:

$$\mathsf{post}_1 = \{\texttt{holds}(a, b), \texttt{electricity\_contract}(b)\}.$$

Suppose the pre-condition of opening a bank account is that the customer c is eligible for a bank account in the UK and holds a proof of address. Moreover, suppose that, if a letter from the employer is available, then the bank account comes with a credit card, otherwise not. This service can be formalised by the service description S_2, which has an empty set of occlusions and the following pre- and post-conditions:

$$\begin{aligned}\mathsf{pre}_2 = \;& \{\texttt{Eligible\_bank}(a), \exists\texttt{holds.Proof\_address}(a)\}\\ \mathsf{post}_2 = \;& \{\texttt{holds}(a, c),\\ & \exists\texttt{holds.Letter}(a)/\texttt{B\_acc\_credit}(c),\\ & \neg\exists\texttt{holds.letter}(a)/\texttt{B\_acc\_no\_credit}(c)\}\end{aligned}$$

Suppose that one can apply for child benefit in the UK if one has a child and a bank account. The service S_3 that offers this application then has the following pre- and post-conditions, and again an empty set of occlusions:

$$\begin{aligned}\mathsf{pre}_3 = \;& \{\texttt{parent\_of}(a, d), \exists\texttt{holds.B\_acc}(a)\}\\ \mathsf{post}_3 = \;& \{\texttt{receives\_c\_benef\_for}(a, d)\}\end{aligned}$$

The meaning of the concepts used in S_1, S_2, and S_3 are defined in the following acyclic TBox $\mathcal{T}$:

$$\begin{aligned}\mathcal{T} = \{\texttt{Eligible\_bank} \;&\equiv\; \exists\texttt{permanent\_resident.}\{\texttt{UK}\},\\ \texttt{Proof\_address} \;&\equiv\; \texttt{Electricity\_contract},\\ \texttt{B\_acc} \;&\equiv\; \texttt{B\_acc\_credit} \sqcup \texttt{B\_acc\_no\_credit}\}\end{aligned}$$

To define the semantics of services, we must first 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 $\mathcal{I}'$. Our definition follows the possible models approach (PMA) initially proposed in [159] and further elaborated e.g. in [160, 158, 48, 71]. Equivalently, we could have translated description logic into first-order logic and then define executability and projection within Reiter's framework for reasoning about deterministic actions [128]. We discuss this approach in Section 9.2. 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 103 (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

- $((A^{\mathcal{I}} \triangledown A^{\mathcal{I}'}) \setminus \{a^{\mathcal{I}} \mid A(a) \in \mathsf{occ}\}) \subseteq A^{\mathcal{I}} \triangledown A^{\mathcal{I}''}$;
- $((s^{\mathcal{I}} \triangledown s^{\mathcal{I}'}) \setminus \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid s(a, b) \in \mathsf{occ}\}) \subseteq s^{\mathcal{I}} \triangledown s^{\mathcal{I}''}$.

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 aber preceq sieht anders aus $\preccurlyeq_{\mathcal{I},S}$ instead of $\preccurlyeq_{\mathcal{I},S,\emptyset}$. $\triangle$

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 104 (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 $\mathcal{I}'$ ($\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$) 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 $\mathcal{I}'$ ($\mathcal{I} \Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_k} \mathcal{I}'$) 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 $\mathcal{I}_{i-1} \Rightarrow^{\mathcal{T}}_{S_i} \mathcal{I}_i$ for $1 \leq i \leq k$. If $\mathcal{T}$ is empty, we write $\Rightarrow_{S_1,\ldots,S_k}$ instead of $\Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_k}$. $\triangle$

Note that this definition does not check whether the service is indeed executable, i.e., whether the pre-conditions are satisfied. It just says what the result of applying the service is, irrespective of whether it is executable or not.

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 $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$. First note that there are indeed cases where there is no successor model $\mathcal{I}'$. In this case, we say that the service is *inconsistent with* $\mathcal{I}$. It is easy to see that this is the case iff there are post-conditions $\varphi_1/\psi, \varphi_2/\neg\psi \in \mathsf{post}$ such that both φ_1 and φ_2 are satisfied in $\mathcal{I}$. Second, assume that S is consistent with $\mathcal{I}$. The fact that there is exactly one model $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$ is an easy consequence of the next lemma, whose proof we leave as an easy exercise.

Lemma 105. *Let $\mathcal{T}$ be an acyclic TBox, $S = (\mathsf{pre}, \emptyset, \mathsf{post})$ a service for $\mathcal{T}$, and $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$ for models $\mathcal{I}, \mathcal{I}'$ of $\mathcal{T}$. If A is a primitive concept and s a role name, then*

$$A^{\mathcal{I}'} := \big(A^{\mathcal{I}} \cup \{b^{\mathcal{I}} \mid \varphi/A(b) \in \mathsf{post} \text{ and } \mathcal{I} \models \varphi\}\big) \setminus \{b^{\mathcal{I}} \mid \varphi/\neg A(b) \in \mathsf{post} \text{ and } \mathcal{I} \models \varphi\},$$

$$s^{\mathcal{I}'} := \big(s^{\mathcal{I}} \cup \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \varphi/s(a,b) \in \mathsf{post} \text{ and } \mathcal{I} \models \varphi\}\big) \setminus \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \varphi/\neg s(a,b) \in \mathsf{post} \text{ and } \mathcal{I} \models \varphi\}.$$

Since the interpretation of the defined concepts is uniquely determined by the interpretation of the primitive concepts and the role names, it follows that there cannot exist more than one $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$.

In principle, we could have started with this more transparent definition of the relation $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S} \mathcal{I}'$ (with some adaptations to deal with occlusions). However, in Section 9.5 we will discuss possible extensions of our approach: for example, to cyclic TBoxes or post-conditions φ/ψ with more complex ABox assertions ψ. In these cases, services are no longer deterministic, and thus the above lemma does not hold. The PMA approach even yields a well-defined semantics for these services (though not necessarily a satisfactory one).

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 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. [128]. In our setting, they can formally be defined as follows:

Definition 106 (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 conditions are true in all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$:
  - $\mathcal{I} \models \mathsf{pre}_1$ and
  - for all i with $1 \leq i < k$ and all interpretations $\mathcal{I}'$ with $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_i} \mathcal{I}'$, 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 $\mathcal{I} \Rightarrow^{\mathcal{T}}_{S_1,\ldots,S_k} \mathcal{I}'$, we have $\mathcal{I}' \models \varphi$.

If $\mathcal{T}$ is empty, we simply drop the phrase "w.r.t. $\mathcal{T}$" instead of writing "w.r.t. the empty TBox $\emptyset$". △

Note that executability alone does not guarantee that we cannot get stuck while executing a composite service. It may also happen that the service to be applied is inconsistent with the current interpretation. 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 $\mathcal{T}$. Summing up, to achieve an effect φ (an ABox assertion) starting from a world description $\mathcal{A}$ and given a TBox $\mathcal{T}$, we need a service $S_1, \ldots, S_k$ such that $S_1, \ldots, S_k$ is executable in $\mathcal{A}$ w.r.t $\mathcal{T}$, S_i is consistent with $\mathcal{T}$ for $1 \leq i \leq k$, and φ is a consequence of applying $S_1, \ldots, S_k$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$.

We do not view consistency with the considered TBox $\mathcal{T}$ as a reasoning task, but rather as a condition that we generally expect to be satisfied by all well-formed services. Still, we should be able to decide whether a service is consistent with a TBox. This can be done by a reduction to standard DL reasoning: given the characterization of consistency *with a model* stated above Lemma 105, it is not difficult to see that an atomic service S with post-conditions post_i is consistent with a TBox $\mathcal{T}$ iff $\{\varphi_1/\psi, \varphi_2/\neg\psi\} \subseteq \mathsf{post}_i$ implies that the ABox $\{\varphi_1, \varphi_2\}$ is inconsistent w.r.t. $\mathcal{T}$.

In our example, all three services are consistent with $\mathcal{T}$. Given the ABox

$$\mathcal{A} = \{\texttt{parent}(a, d), \texttt{permanent\_resident}(a, \texttt{UK})\},$$

it is easily checked that the composite service $S = S_1, S_2, S_3$ is executable, and that $\texttt{receives\_c\_benef\_for}(a, d)$ is a consequence of applying S in $\mathcal{A}$ w.r.t. $\mathcal{T}$. Note that the presence of the TBox is crucial for this result.

The main aim of this paper is to show how the two reasoning tasks executability and projection can be computed, and how their complexity depends on the description logic used within our framework. There is one particularly simple case: for atomic services S, computing executability boils down to standard DL reasoning: S is executable in $\mathcal{A}$ w.r.t. $\mathcal{T}$ iff $\mathcal{A} \cup \{\neg\varphi\}$ is inconsistent w.r.t. $\mathcal{T}$ for all $\varphi \in \mathsf{pre}$. Executability for composite services is less trivial, and the same holds for projection of both atomic and

composite services. We show now that the two reasoning services can be mutually polynomially reduced to each other. This allows us to concentrate on projection when proving decidability and complexity results. The following lemma is proved in [10]:

Lemma 107. *Executability and projection can be reduced to each other in polynomial time.*

Relationship with SitCalc

We have chosen a possible models approach to define the effects of our services. More established and widely used in the reasoning about action community is the situation calculus [128]. In contrast to the PMA, the situation calculus uses an axiomatic approach to define the effects of actions. However, if we consider services without occlusions, then our approach can be seen as an instance of the situation calculus.

Suppose an ABox $\mathcal{A}$, an acyclic TBox $\mathcal{T}$, and a composite service $S_1, \ldots, S_k$ are given. First, we can get rid of the TBox by expanding it and then replacing in $\mathcal{A}$ and the services $S_1, \ldots, S_k$ the defined concepts with their definitions.[28] Consider now the simple description of the relation $\Rightarrow_S^{\mathcal{T}}$ given in Lemma 105. By taking the standard translation of $\mathcal{ALCQIO}$ into first-order logic [8], we can easily translate this description into *action pre-conditions* and *successor state axioms* in the sense of [128]. In this setting, primitive concepts and role names are regarded as fluents. We take the first-order translation of the ABox as the initial state, and then we can show that our notions of executability and projection are instances of Reiter's definitions (see [11] for details).

The translation of our approach into a situation calculus axiomatization à la Reiter shows that our formalism is firmly based on research on reasoning about action. However, this does not mean that the inference problems introduced above can be solved using an implemented system for reasoning about action, such as GOLOG [96]. In fact, in Reiter's approach, *regression* [128] is used to solve the executability and the projection problem. However, when applied to (the translation of) our services, regression yields a standard first-order theory, which is not in the scope of what GOLOG can handle without calling a general first-order theorem prover. Thus, the translation into situation calculus does not directly provide us with decidability or complexity results for our reasoning problems.

9.3 Decision Procedures

We develop reasoning procedures for the reasoning services introduced in Section 9.2 and analyze the computational complexity of executability and projection of different fragments of $\mathcal{ALCQIO}$. Throughout this section, we assume that all services are consistent with their TBox, and that TBoxes are acyclic.

By Lemma 107, 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 [128]: 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 $\mathcal{I}$. Concerning the standard reasoning tasks, we consider two options:

Firstly, we show that the theory we obtain can again be expressed by a description logic 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. We prove lower complexity bounds for projection showing that the increase in complexity that is sometimes obtained by introducing nominals cannot be avoided.

[28] Alternatively, we could handle the TBox as state constraints.

Secondly, we show that we can express the resulting theory in C^2, the two-variable fragment of first-order logic extended with counting quantifiers. This way, projection is reduced to satisfiability in C^2. We obtain a simpler reduction, but less sharp complexity results since satisfiability in C^2 is NEXPTIME-complete [121, 124], and thus quite costly from a computational perspective. However, there are two exceptional cases where we obtain a tight upper bound using the second translation, but not the first: $\mathcal{ALCQI}$ and $\mathcal{ALCQIO}$ with numbers in number restrictions coded in binary, i.e., the size of $(\geqslant n\ r\ C)$ and $(\leqslant n\ r\ C)$ is assumed to be $\log(n) + 1$ plus the size of C.

The following results are proved in this section:

Theorem 108. *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 description logic $\mathcal{L}$ coincides with the complexity of inconsistency of ABoxes in $\mathcal{LO}$, the extension of $\mathcal{L}$ with nominals.

Reduction to DL Reasoning

We reduce projection in fragments $\mathcal{L}$ of $\mathcal{ALCQIO}$ to ABox (in)consistency in the extension $\mathcal{LO}$ of $\mathcal{L}$ with nominals. Here, we assume unary coding of numbers in number restrictions, i.e., the size of $(\leqslant n\ r\ C)$ and $(\geqslant n\ r\ C)$ is assumed to be $n + 1$ plus the size of C.

Theorem 109. *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 inconsistency in $\mathcal{LO}$ w.r.t. acyclic TBoxes.*

Let $\mathcal{L}$ be one of the languages listed in Theorem 109, and let $\mathcal{A}$ be an ABox, $S_1, \ldots, S_n$ a composite service with $S_i = (\mathsf{pre}_i, \mathsf{occ}_i, \mathsf{post}_i)$, $\mathcal{T}$ an acyclic TBox, and φ_0 an assertion, all formulated in $\mathcal{L}$. We are interested in deciding whether φ_0 is a consequence of applying $S_1, \ldots, S_n$ in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$. Without loss of generality, we assume that φ_0 is of the form $A_0(a_0)$, for a concept name A_0:

1. Assertions $r(a, b)$ and $\neg r(a, b)$ can be replaced with $(\exists r.\{b\})(a)$ and $(\forall r.\neg\{b\})(a)$, respectively. This presupposes nominals, but nominals will be used in our reduction, anyway.
2. If $\varphi = C(a)$ with C not a concept name, we add a concept definition $A_0 \equiv C$ to the TBox $\mathcal{T}$, and then consider $\varphi = A_0(a)$.

In the following, we call $\mathcal{A}$, $\mathcal{T}$, $S_1, \ldots, S_n$, and φ_0 *the input*. We devise a reduction ABox $\mathcal{A}_{\mathsf{red}}$, an (acyclic) reduction TBox $\mathcal{T}_{\mathsf{red}}$, and a reduction assertion φ_{red} such that

φ_0 is a consequence of applying $S_1, \ldots, S_n$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$ iff $\mathcal{A}_{\mathsf{red}}$ is inconsistent w.r.t. $\mathcal{T}_{\mathsf{red}}$.

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 $\mathcal{I}_0, \ldots, \mathcal{I}_n$ obtained by applying $S_1, \ldots, S_n$ in $\mathcal{A}$ (and *all* such sequences are encoded by reduction models). To ensure this, we use the following intuitions:

- The reduction ABox states that (i) the "$\mathcal{I}_0$-part" of a reduction model $\mathcal{I}$ is a model of $\mathcal{A}$, and that (ii) the $\mathcal{I}_i$-part of $\mathcal{I}$ satisfies the post-conditions post_i, for $1 \leq i \leq n$.
- The reduction TBox states that the $\mathcal{I}_i$-part of $\mathcal{I}$ is a model of $\mathcal{T}$, for each $i \leq n$.
- We need to describe the law of inertia, i.e., the fact that we want to minimize the changes that are performed when applying a service. This task is split among the reduction ABox and TBox.

To understand the splitting mentioned in the third item, it is important to distinguish two kinds of elements in interpretations: we call an element $d \in \Delta^{\mathcal{I}}$ *named* if $a^{\mathcal{I}} = d$ for some individual a used in the input, and *unnamed* otherwise. Intuitively, the minimization of changes on named elements can be described in a direct way through the ABox $\mathcal{A}_{\mathsf{red}}$, while the minimization of changes on unnamed elements is achieved through a suitable encoding of $\mathcal{T}$ in $\mathcal{T}_{\mathsf{red}}$. Indeed, minimizing changes on unnamed elements boils down to enforcing that changes in concept (non)membership and role (non)membership involving (at least) one unnamed domain element *never* occur: due to the restriction to primitive concept names in post-conditions, our services are not expressive enough to enforce such changes.

In the reduction, we use the following concept names, role names, and individual names:

- The smallest set that contains all concepts appearing in the input and is closed under taking subconcepts is denoted with Sub. For every $C \in \mathsf{Sub}$ and every $i \leq n$, we introduce a concept name $T_C^{(i)}$. It will be ensured by the TBox $\mathcal{T}_{\mathsf{red}}$ that the concept name $T_C^{(i)}$ stands for the interpretation of C in the i-th interpretation $\mathcal{I}_i$.
- We use a concept name $A^{(i)}$ for every primitive concept name A used in the input and every $i \leq n$. Intuitively, $A^{(i)}$ represents the interpretation of the concept name A in $\mathcal{I}_i$, *but only with respect to the named domain elements*. Since concept membership of unnamed elements never changes, the "unnamed part" of the interpretation of A in $\mathcal{I}_i$ can be found in $A^{(0)}$, for *any* $i \leq n$.
- We use a role name $r^{(i)}$ for every role name r used in the input and every $i \leq n$. Similarly to concept names, $r^{(i)}$ stands for the interpretation of r in $\mathcal{I}_i$ *but only concerning those role relationships where both involved domain elements are named*. All other role relationships never change and are stored in $r^{(0)}$.
- We use a concept name N to denote named elements of interpretations.
- The set of individual names used in the input is denoted with Obj. For every $a \in \mathsf{Obj}$, we introduce an auxiliary role name r_a.
- Finally, we use an auxiliary individual name $a_{\mathsf{help}} \notin \mathsf{Obj}$.

The reduction TBox $\mathcal{T}_{\mathsf{red}}$ consists of several components. The first component simply states that N denotes exactly the named domain elements:

$$\mathcal{T}_N := \Big\{ N \equiv \bigsqcup_{a \in \mathsf{Obj}} \{a\} \Big\}.$$

The second component $\mathcal{T}_{\mathsf{sub}}$ contains one concept definition for every $i \leq n$ and every concept $C \in \mathsf{Sub}$ that is not a defined concept name in $\mathcal{T}$. These concept definitions ensure that $T_C^{(i)}$ stands for the interpretation of C in $\mathcal{I}_i$ as desired. Details are given in Figure 17, where $r^{-(i)}$ denotes $(r^{(i)})^-$ in the concept definitions for number restrictions. The first concept definition reflects the fact that concept names $A^{(i)}$ only represent the extension of A in $\mathcal{I}_i$ *for named domain elements*. To get $T_A^{(i)}$, the full extension of A in $\mathcal{I}_i$, we use $A^{(i)}$ for named elements and $A^{(0)}$ for unnamed ones. A similar splitting

$$
\begin{aligned}
T_A^{(i)} &\equiv (N \sqcap A^{(i)}) \sqcup (\neg N \sqcap A^{(0)}) \quad A \text{ primitive in } \mathcal{T} \\
T_{\neg C}^{(i)} &\equiv \neg T_C^{(i)} \\
T_{C \sqcap D}^{(i)} &\equiv T_C^{(i)} \sqcap T_D^{(i)} \\
T_{C \sqcup D}^{(i)} &\equiv T_C^{(i)} \sqcup T_D^{(i)} \\
T_{(\geqslant m\ r\ C)}^{(i)} &\equiv \Big(N \sqcap \bigsqcup_{0 \leq j \leq m} \big((\geqslant j\ r^{(i)}\ (N \sqcap T_C^{(i)})) \sqcap (\geqslant (m-j)\ r^{(0)}\ (\neg N \sqcap T_C^{(i)}))\big)\Big) \\
&\quad \sqcup \Big(\neg N \sqcap (\geqslant m\ r^{(0)}\ T_C^{(i)})\Big) \\
T_{(\leqslant m\ r\ C)}^{(i)} &\equiv \Big(N \sqcap \bigsqcup_{0 \leq j \leq m} \big((\leqslant j\ r^{(i)}\ (N \sqcap T_C^{(i)})) \sqcap (\leqslant (m-j)\ r^{(0)}\ (\neg N \sqcap T_C^{(i)}))\big)\Big) \\
&\quad \sqcup \Big(\neg N \sqcap (\leqslant m\ r^{(0)}\ T_C^{(i)})\Big)
\end{aligned}
$$

Figure 17: The TBox $\mathcal{T}_{\mathsf{sub}}$.

of role relationships into a named part and an unnamed part is reflected in the translation of number restrictions given in the last two concept definitions.

Now we can assemble the reduction TBox $\mathcal{T}_{\mathsf{red}}$:

$$\mathcal{T}_{\mathsf{red}} \ := \ \mathcal{T}_{\mathsf{sub}} \cup \mathcal{T}_N \ \cup \{T_A^{(i)} \equiv T_E^{(i)} \mid A \equiv E \in \mathcal{T},\ i \leq n\}.$$

The last summand of $\mathcal{T}_{\mathsf{red}}$ ensures that all definitions from the input TBox $\mathcal{T}$ are satisfied by $\mathcal{I}_0, \ldots, \mathcal{I}_n$.

The reduction ABox $\mathcal{A}_{\mathsf{red}}$ also consists of several components. The first component ensures that, for each individual a occurring in the input, the auxiliary role r_a connects each individual (including a_{help}) with a, and only with a. This construction will simplify the definition of the other components of $\mathcal{A}_{\mathsf{red}}$:

$$\mathcal{A}_{\mathsf{aux}} := \big\{a : \big(\exists r_b.\{b\} \sqcap \forall r_b.\{b\}\big) | a \in \mathsf{Obj} \cup \{a_{\mathsf{help}}\},\ b \in \mathsf{Obj}\big\}.$$

To continue, we first introduce the following abbreviations, for $i \leq n$:

$$
\begin{aligned}
\mathsf{p}_i(C(a)) &:= \forall r_a.T_C^{(i)} \\
\mathsf{p}_i(r(a,b)) &:= \forall r_a.\exists r^{(i)}.\{b\} \\
\mathsf{p}_i(\neg r(a,b)) &:= \forall r_a.\forall r^{(i)}.\neg\{b\}.
\end{aligned}
$$

The next component of $\mathcal{A}_{\mathsf{red}}$ formalizes satisfaction of the post-conditions. Note that its formulation relies on $\mathcal{A}_{\mathsf{aux}}$. For $1 \leq i \leq n$, we define

$$\mathcal{A}_{\mathsf{post}}^{(i)} := \big\{a_{\mathsf{help}} : \big(\mathsf{p}_{i-1}(\varphi) \rightarrow \mathsf{p}_i(\psi)\big) \mid \varphi/\psi \in \mathsf{post}_i\big\}.$$

We now formalize the minimization of changes on named elements. For $1 \leq i \leq n$ the ABox $\mathcal{A}_{\mathrm{min}}^{(i)}$ contains

– the following assertions for every $a \in \mathsf{Obj}$ and every primitive concept name A with $A(a) \notin \mathsf{occ}_i$:

$$
\begin{aligned}
&a{:}\Big(\big(A^{(i-1)} \sqcap \bigsqcap_{\varphi/\neg A(a) \in \mathsf{post}_i} \neg \mathsf{p}_{i-1}(\varphi)\big) \rightarrow A^{(i)}\Big) \\
&a{:}\Big(\big(\neg A^{(i-1)} \sqcap \bigsqcap_{\varphi/A(a) \in \mathsf{post}_i} \neg \mathsf{p}_{i-1}(\varphi)\big) \rightarrow \neg A^{(i)}\Big);
\end{aligned}
$$

– the following assertions for all $a, b \in \mathsf{Obj}$ and every role name r with $r(a,b) \notin \mathsf{occ}_i$:

$$a{:}\Big(\big(\exists r^{(i-1)}.\{b\} \sqcap \bigsqcap_{\varphi/\neg r(a,b)\in \mathsf{post}_i} \neg \mathsf{p}_{i-1}(\varphi)\big) \rightarrow \exists r^{(i)}.\{b\}\Big)$$
$$a{:}\Big(\big(\forall r^{(i-1)}.\neg\{b\} \sqcap \bigsqcap_{\varphi/r(a,b)\in \mathsf{post}_i} \neg \mathsf{p}_{i-1}(\varphi)\big) \rightarrow \forall r^{(i)}.\neg\{b\}\Big).$$

The ABox $\mathcal{A}_{\mathsf{ini}}$ ensures that the first interpretation of the encoded sequence is a model of the input ABox $\mathcal{A}$:

$$\begin{aligned}\mathcal{A}_{\mathsf{ini}} \;:=\; & \{T_C^{(0)}(a) \mid C(a) \in \mathcal{A}\} \cup \\ & \{r^{(0)}(a,b) \mid r(a,b) \in \mathcal{A}\} \cup \\ & \{\neg r^{(0)}(a,b) \mid \neg r(a,b) \in \mathcal{A}\}.\end{aligned}$$

We can now assemble $\mathcal{A}_{\mathsf{red}}$:

$$\begin{aligned}\mathcal{A}_{\mathsf{red}} \;:=\; & \mathcal{A}_{\mathsf{ini}} \cup \mathcal{A}_{\mathsf{aux}} \cup \\ & \mathcal{A}^{(1)}_{\mathsf{post}} \cup \cdots \cup \mathcal{A}^{(n)}_{\mathsf{post}} \cup \\ & \mathcal{A}^{(1)}_{\mathsf{min}} \cup \cdots \cup \mathcal{A}^{(n)}_{\mathsf{min}} \cup \\ & \{\neg T^{(n)}_{A_0}(a_0)\}.\end{aligned}$$

The proof of the following lemma can be found in [11].

Lemma 110. *$A_0(a_0)$ is a consequence of applying $S_1, \ldots, S_n$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$ iff $\mathcal{A}_{\mathsf{red}}$ is inconsistent w.r.t. $\mathcal{T}_{\mathsf{red}}$.*

Since the size of $\mathcal{A}_{\mathsf{red}}$, $\mathcal{T}_{\mathsf{red}}$, and φ_{red} are clearly polynomial in the size of the input (recall that we assume unary coding of numbers in number restrictions), Lemma 110 immediately yields Theorem 109. Thus, for the DLs $\mathcal{L}$ considered in Theorem 109, upper complexity bounds for ABox inconsistency in $\mathcal{LO}$ carry over to projection in $\mathcal{L}$. Many such upper bounds are available from the literature. Indeed, there is only one case where we cannot draw upon existing results: the complexity of ABox consistency in $\mathcal{ALCQO}$ w.r.t. acyclic TBoxes. For the sake of completeness, we prove that this problem is PSPACE-complete in Appendix A of [11]. Lower complexity bounds carry over from ABox inconsistency in a DL $\mathcal{L}$ to projection in the same DL: $\mathcal{A}$ is not consistent w.r.t. $\mathcal{T}$ iff $a : \bot$ 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 $\mathcal{L}$ that allow for nominals or where the addition of nominals does *not* increase the complexity of reasoning. Using the known lower complexity bounds for ABox (in)consinstency [135, 4, 144, 121], we obtain the following Corollary:

Corollary 111. *Executability and projection w.r.t. acyclic TBoxes are*

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

Points 1, 4, and 5 presuppose that numbers in number restrictions are coded in unary.

In Section 9.4, we prove matching lower bounds for Points 2 and 4 of Corollary 108.

Reduction to C2

Alternatively to reducing to standard DL reasoning, we can reduce projection to satisfiability in C^2. This yields a simpler translation and a co-NEXPTIME upper bound for projection in $\mathcal{ALCQI}$ and $\mathcal{ALCQIO}$ with numbers in number restrictions coded in binary—in contrast to the reduction given in the previous section which requires unary coding to yield co-NEXPTIME upper bounds (otherwise, the last two lines of Figure 17 yield an exponential blow-up). However, we cannot get any PSPACE or EXPTIME upper bounds from the C^2-translation since satisfiability in C^2 is NEXPTIME-complete [121, 124].

The intuitions underlying the reduction to C^2 are very similar to those given in the previous section, apart from one significant simplification: since C^2 is more expressive than $\mathcal{ALCQIO}$, it is not necessary to split the interpretations of concept and role names into a named part and an unnamed part. Full details are given in [11]. We obtain the following result:

Theorem 112. *Projection of composite services formulated in $\mathcal{ALCQIO}$ can be polynomially reduced to satisfiability in C^2.*

Together with the reduction from executability to projection, this yields the following result, which sharpens Points 4 and 5 of Corollary 111 to cover also the case of binary coding of numbers inside number restrictions.

Corollary 113. *Executability and projection w.r.t. acyclic TBoxes are in co-*NEXPTIME *for $\mathcal{ALCQIO}$ even if the numbers in number restrictions are coded in binary.*

A matching lower bound for $\mathcal{ALCQIO}$ is obtained from Point 5 of Corollary 111. As shown in the following subsection, Corollary 113 also yields a tight upper bound for the fragment $\mathcal{ALCQI}$ of $\mathcal{ALCQIO}$.

9.4 Hardness Results

We show that the upper bounds for executability and projection obtained in the previous two subsections cannot be improved. In Section 9.3, we have already obtained matching lower bounds for DLs $\mathcal{L}$ where the complexity of ABox inconsistency coincides in $\mathcal{L}$ and $\mathcal{LO}$ ($\mathcal{L}$'s extension with nominals). It thus remains 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. By Lemma 107, these bounds carry over to executability, thus matching Points 2 and 4 of Corollary 108. The results established in this subsection 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.[29] Observe that the coding of numbers, i.e. unary vs. binary, is not an issue in $\mathcal{ALCFIO}$ and $\mathcal{ALCFI}$, and thus a lower bound for projection in $\mathcal{ALCFI}$ implies the same bound for projection in $\mathcal{ALCQI}$ with unary coding of numbers. Our aim is to prove the following.

Theorem 114. *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,*

[29] We admit the number 0 to preserve the abbreviation $\forall r.C$ that stands for $(\leqslant 0\ r\ \neg C)$.

- *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 $\mathcal{ALCFI}$.

For the proof of Theorem 114, let $\mathcal{L} \in \{\mathcal{ALCIO}, \mathcal{ALCFIO}\}$ and C an $\mathcal{L}$-concept whose unsatisfiability is to be decided. For simplicity, we assume that C contains only a single nominal $\{n\}$. This can be done w.l.o.g. since the complexity of unsatisfiability in $\mathcal{ALCIO}$ (resp. $\mathcal{ALCFIO}$) is already EXPTIME-hard (resp. co-NEXPTIME-hard) if only a single nominal is available and TBoxes are not admitted [4, 144, 143]. For the reduction, we reserve a new concept name O and a role name u that do not occur in C. Let

$$\mathsf{rol}(C) := \{r, r^- \mid r \in \mathsf{N_R} \text{ used in } C\}$$

and let $C[O/\{n\}]$ denote the result of replacing each occurrence of the nominal $\{n\}$ in C with the concept name O. We define an ABox $\mathcal{A}$, an atomic service $S = (\emptyset, \emptyset, \mathsf{post}_S)$, and a concept D_C as follows:

$$\begin{aligned} \mathcal{A}_C &:= \{a : (\neg O \sqcap \forall u.\neg O \sqcap \forall u. \bigsqcap_{r \in \mathsf{rol}(C)} \forall r.\exists u^-.\neg O)\} \\ \mathsf{post}_S &:= O(a) \\ D_C &:= \exists u.C[O/\{n\}] \sqcap (\forall u. \bigsqcap_{r \in \mathsf{rol}(C)} \forall r.\forall u^-.O) \end{aligned}$$

Let $\mathcal{I}$ and $\mathcal{I}'$ be models witnessing that $\neg D_C(a)$ is *not* a consequence of S, i.e., $\mathcal{I} \models \mathcal{A}_C$, $\mathcal{I} \Rightarrow_S \mathcal{I}'$, and $\mathcal{I}' \models D_C(a)$. The reduction rests on the following ideas:

- By the first conjunct of (the concept in) $\mathcal{A}_C$, the post-condition, and Lemma 105, the only difference between $\mathcal{I}$ and $\mathcal{I}'$ is that $a^{\mathcal{I}} = a^{\mathcal{I}'} \in O^{\mathcal{I}'} \setminus O^{\mathcal{I}}$;
- By the first conjunct of (the concept in) $\mathcal{A}_C$ and the post-condition, the only difference between $\mathcal{I}$ and $\mathcal{I}'$ is that $a^{\mathcal{I}} = a^{\mathcal{I}'} \in O^{\mathcal{I}'} \setminus O^{\mathcal{I}}$;
- Using the first and third conjunct of $\mathcal{A}_C$ together with the post-condition and the second conjunct of D_C, it can be shown that $(a^{\mathcal{I}}, x) \in u^{\mathcal{I}} = u^{\mathcal{I}'}$ for each x from the relevant part rel of $\Delta^{\mathcal{I}}$, where rel is defined as the smallest set that contains $a^{\mathcal{I}}$ and is closed under taking successors for the roles from $\mathsf{rol}(C)$;
- Thus, the second conjunct of $\mathcal{A}_C$ ensures that $O^{\mathcal{I}} \cap \mathsf{rel} = \emptyset$ and $O^{\mathcal{I}'} \cap \mathsf{rel} = \{a^{\mathcal{I}}\}$.
- Due to the first conjunct of D_C, $C[O/\{n\}]$ is satisfied in the relevant part of $\mathcal{I}'$. By the previous item, the concept name O behaves like a nominal.

In [11], we prove the following lemma, which immediately yields Theorem 114.

Lemma 115. *The following statements are equivalent:*

1. *C is satisfiable.*
2. *$\neg D_C(a)$ is not a consequence of applying S in $\mathcal{A}_C$.*
3. *the composite service $S, (\{\neg D_C(a)\}, \emptyset, \emptyset)$ is not executable in $\mathcal{A}_C$.*

9.5 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, which are available in OWL-DL;
2. we only allow for acyclic TBoxes rather than arbitrary (also cyclic) ones or even so-called general concept inclusions (GCIs), which are also available in OWL-DL;
3. in post-conditions $\varphi/C(a)$, we require C to be a primitive concept or its negation, rather than admitting arbitrary, complex concepts.

The purpose of this section is to provide a justification for these restrictions: we show that removing the first restriction leads to *semantic problems*, while removing the second and third restriction leads to *both semantic and computational problems*.

Transitive Roles

Transitive roles are offered by most modern DL systems [79, 66], and also by the ontology languages OWL, DAML+OIL, and OIL [83, 81, 49]. They are added to $\mathcal{ALCQIO}$ by reserving a subset of roles $\mathsf{N_{tR}}$ of $\mathsf{N_R}$ such that all $r \in \mathsf{N_{tR}}$ are interpreted as transitive relations $r^{\mathcal{I}}$ in all models $\mathcal{I}$. We show that admitting the use of transitive roles in post-conditions yields semantic problems.

By Lemma 105, services without occlusions $S = (\mathsf{pre}, \emptyset, \mathsf{post})$ are deterministic in the sense that $\mathcal{I} \Rightarrow_S^{\mathcal{I}} \mathcal{I}'$, and $\mathcal{I} \Rightarrow_S^{\mathcal{I}} \mathcal{I}''$ implies $\mathcal{I}' = \mathcal{I}''$. This is not any more the case for services referring to transitive roles: consider the service $S = (\emptyset, \emptyset, \{\mathsf{has\text{-}part}(\mathsf{car}, \mathsf{engine})\})$ that adds an engine to a car. Let has-part be a transitive role and take the model

$$\begin{aligned} \Delta^{\mathcal{I}} &:= \{\mathsf{car}, \mathsf{engine}, \mathsf{valve}\} \\ \mathsf{has\text{-}part}^{\mathcal{I}} &:= \{(\mathsf{engine}, \mathsf{valve})\} \\ z^{\mathcal{I}} &:= z \text{ for } z \in \Delta^{\mathcal{I}}. \end{aligned}$$

Then we have both $\mathcal{I} \Rightarrow_S \mathcal{I}'$ and $\mathcal{I} \Rightarrow_S \mathcal{I}''$, where $\mathcal{I}'$ is obtained from $\mathcal{I}$ by setting

$$\mathsf{has\text{-}part}^{\mathcal{I}'} := \{(\mathsf{car}, \mathsf{engine}), (\mathsf{engine}, \mathsf{valve}), (\mathsf{car}, \mathsf{valve})\}$$

and $\mathcal{I}''$ is obtained from $\mathcal{I}$ by setting

$$\mathsf{has\text{-}part}^{\mathcal{I}''} := \{(\mathsf{car}, \mathsf{engine})\}.$$

Observe that, in $\mathcal{I}''$, the valve is no longer part of the engine since adding only $(\mathsf{car}, \mathsf{engine})$ to $\mathsf{has\text{-}part}^{\mathcal{I}}$ violates the transitivity of has-part. Hence, in contrast to our intuition, $\mathsf{has\text{-}part}(\mathsf{engine}, \mathsf{valve})$ is not a cosequence of applying S in $\{\mathsf{has\text{-}part}(\mathsf{engine}, \mathsf{valve})\}$.

In the area of reasoning about actions, it is well-known that non-determinism of this kind requires extra effort to obtain sensible consequences of action/service executions [99, 141]. Thus, we need a mechanism for eliminating unwanted outcomes or preferring the desired ones. We leave such extensions as future work.

Cyclic TBoxes and GCIs

Assume that we admit arbitrary (also cyclic) TBoxes as defined in Section 8.2. Then semantic problems arise due to a crucial difference between cyclic and acyclic TBoxes: 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. To see this, consider the following example:

$$\begin{aligned} \mathcal{A} &:= \{\mathsf{Dog}(a)\} \\ \mathcal{T} &:= \{\mathsf{Dog} \equiv \exists \mathsf{parent}.\mathsf{Dog}\} \\ \mathsf{post} &:= \{\mathsf{Cat}(b)\} \end{aligned}$$

Then, $\mathsf{Dog}(a)$ is *not* a consequence of applying $S = (\emptyset, \emptyset, \mathsf{post})$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$, as one would intuitively expect. This is due to the following countermodel. Define an interpretation $\mathcal{I}$ as follows:

$$\begin{aligned} \Delta^{\mathcal{I}} &:= \{b\} \cup \{d_0, d_1, d_2, \ldots\} \\ \mathsf{Dog}^{\mathcal{I}} &:= \{d_0, d_1, d_2, \ldots\} \\ \mathsf{Cat}^{\mathcal{I}} &:= \emptyset \\ \mathsf{parent}^{\mathcal{I}} &:= \{(d_i, d_{i+1}) \mid i \in \mathbb{N}\} \\ a^{\mathcal{I}} &:= d_0 \\ b^{\mathcal{I}} &:= b \end{aligned}$$

The interpretation $\mathcal{I}'$ is defined as $\mathcal{I}$, with the exception that $\mathsf{Cat}^{\mathcal{I}'} = \{b\}$ and $\mathsf{Dog}^{\mathcal{I}'} := \emptyset$. Using the fact that Dog is a defined concept and thus not considered in the definition of $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$, it is easy to see that $\mathcal{I} \models \mathcal{A}$, $\mathcal{I} \Rightarrow_S^{\mathcal{T}} \mathcal{I}'$, and $\mathcal{I}' \not\models \mathsf{Dog}(a)$.

There appear to be two possible ways to solve this problem: either include defined concepts in the minimization of changes, i.e., treat them in the definition of $\preccurlyeq_{\mathcal{I},S,\mathcal{T}}$ in the same way as primitive concepts, or use a semantics that regains the "definitorial power" of acyclic TBoxes, namely that an interpretation of the primitive concepts *uniquely* determines the interpretation of defined concepts. The first option is infeasible since minimizing a defined concept A with TBox definition $A \equiv C$ corresponds to minimizing the complex concept C, and it is well-known that even the minimization of arbitrary Boolean concepts (in particular of disjunctions) induces technical problems and counterintuitive results [98]. The second option seems more feasible: if we adopt the least or greatest fixpoint semantics for TBoxes as first proposed by Nebel [116], it is indeed the case that primitive concepts uniquely determine defined concepts. Thus, it may be interesting to analyze services with cyclic TBoxes under fixpoint semantics as future work.

Even more general than admitting cyclic TBoxes is to allow general concept inclusions (GCIs). A *GCI* is an expression $C \sqsubseteq D$, with C and D (possibly complex) concepts. An interpretation $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. As we can rewrite a concept equation $A \equiv C$ as two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$, it should be obvious that (sets of) GCIs strictly generalize (also cyclic) TBoxes. When admitting GCIs in connection with services, we thus run into the same problems as with cyclic TBoxes. 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. However, together with the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$ with C a complex concept, the literal post-condition $\varphi/A(a)$ is equivalent to the complex one $\varphi/C(a)$. Thus, it seems that GCIs cannot be admitted without simultaneously admitting arbitrarily complex concepts in post-conditions. As we will discuss in the following section, this step induces additional semantic problems as well as computational problems.

Complex Concepts in Post-Conditions

Let a *generalized* service be a service where post-conditions are of the form φ/ψ for arbitrary assertions φ and ψ. In other words, ψ is no longer restricted to be a literal over primitive concepts. For simplicity, further assume that occlusions are disallowed and that neither TBoxes nor GCIs are admitted. As we shall discuss in the following, 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.

We start by presenting *semantic problems* induced by complex concepts in post-conditions. Clearly, generalized services such as $S = (\emptyset, \emptyset, \{a : A \sqcup B\})$ are not deterministic and thus introduce similar complications as discussed for transitive roles. However, disjunction is not the only constructor to introduce non-determinism when allowed in post-conditions:

- If a post-condition contains $a : \exists r.A$ and this assertion was not already satisfied before the execution of the service, then the non-determinism lies in the choice of a witness object, i.e., *any* domain element $x \in \Delta^{\mathcal{I}}$ may be chosen to satisfy $(a^{\mathcal{I}}, x) \in r^{\mathcal{I}}$ and $x \in A^{\mathcal{I}}$ after execution of the service.

  The fact that *any* domain element is a potential witness object implies that, e.g., $\mathsf{Female}(\mathsf{mary})$ is not a consequence of applying the service

  $$(\emptyset, \emptyset, \{\mathsf{mary} : \exists \mathsf{has\text{-}child}.\neg\mathsf{Female}\})$$

  in the ABox $\{\mathsf{Female}(\mathsf{mary})\}$.

- If a post-condition contains $a : \forall r.A$ and this assertion was not already satisfied before the execution of the service, we also have a non-deterministic situation: for each object $x \in \Delta^{\mathcal{I}}$ such that $(a^{\mathcal{I}}, x) \in r^{\mathcal{I}}$ and $x \notin A^{\mathcal{I}}$ holds before the execution of the service, we have to decide whether $(a^{\mathcal{J}}, x) \notin r^{\mathcal{J}}$ or $x \in A^{\mathcal{J}}$ should be satisfied after execution of the service.[30]

  Similarly to the existential case, we may obtain surprising results due to the fact that *any* domain element $x \in \Delta^{\mathcal{I}}$ may satisfy $(a^{\mathcal{I}}, x) \in r^{\mathcal{I}}$ and $x \in A^{\mathcal{I}}$ unless explicitly stated otherwise. This means that, e.g., $\mathsf{Filled}(\mathsf{tire2})$ is not a consequence of applying the service

  $$(\emptyset, \emptyset, \{\mathsf{car1}{:}\forall \mathsf{tire}.\mathsf{Filled}\})$$

  in the ABox $\{\mathsf{tire}(\mathsf{car2}, \mathsf{tire2}),\ \neg\mathsf{Filled}(\mathsf{tire2})\}$.

Complex concepts with many nested operators may obviously introduce a rather high degree of non-determinism. While simple non-determinism such as the one introduced by transitive roles or post-conditions $a : C \sqcup D$ may be dealt with in a satisfactory way [99, 141], none of the mainstream action formalisms allows arbitrary formulas in post-conditions. Indeed, most formalisms such as the basic situation calculus restrict themselves to literals in post-conditions [128, 140]—just as our non-generalized services do.

[30] There may even be cases where it is intended that both conditions are satisfied after service execution; this is, however, not justified by the PMA semantics of generalized services.

We continue by presenting *computational problems* which occur in the presence of complex concepts in post-conditions of actions. Namely, executability and projection for generalized services easily become undecidable. To illustrate this, we prove undecidability of these reasoning tasks for the DL $\mathcal{ALCFI}$ that has been introduced in Section 9.4.[31] This result should be contrasted with the fact that, by Theorem 108, reasoning with non-generalized services is decidable even for powerful extensions of $\mathcal{ALCFI}$. Note that $\mathcal{ALCFI}$ may be viewed as a fragment of OWL light, the weakest OWL dialect [82].

Theorem 116. *There exists a generalized atomic service S and an ABox $\mathcal{A}$ formulated in $\mathcal{ALCFI}$ such that the following problems are undecidable: given a concept C,*

- *decide whether the assertion $C(a)$ is a consequence of applying S in $\mathcal{A}$;*
- *decide whether the composite service S, S' is executable in $\mathcal{A}$, where $S' = (\{C(a)\}, \emptyset, \emptyset)$.*

The proof of Theorem 116 is by reduction of the well-known undecidable domino problem [20] to non-consequence and non-executability. For details, see [11].

9.6 Conclusion

The main technical result presented in this section is that 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 description logics $\mathcal{L}$. 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 can 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. Results on ABox updates are presented in Section 8. Second, the expressiveness of 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.

[31] Recall that $\mathcal{ALCFI}$ is obtained from $\mathcal{ALCQI}$ by limiting numbers occurring in number restrictions to $\{0, 1\}$.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[2] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.

[3] C. Areces, P. Blackburn, B. Martinez Hernandez, and M. Marx. Handling boolean aboxes. In *Proceedings of the 2003 International Workshop on Description Logics*, 2003.

[4] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in Lecture Notes in Computer Science, pages 307–321. Springer-Verlag, 1999.

[5] C. Areces and M. de Rijke. From description logics to hybrid logics, and back. In Frank Wolter, Heinrich Wansing, Maarten de Rijke, and Michael Zakharyaschev, editors, *Advances in Modal Logics Volume 3*. CSLI Publications, Stanford, CA, USA, 2001.

[6] F. Baader, R. Bernardi, D. Calvanese, A. Calì, B. Cuenca Grau, M. Garcia, G. de Giacomo, A. Kaplunova, O. Kutz, D. Lembo, M. Lenzerini, L. Lubyte, C. Lutz, M. Milicic, R. Möller, B. Parsia, R. Rosati, U. Sattler, B. Sertkaya, S. Tessaris, C. Thorne, and A.-Y. Turhan. Techniques for ontology design and maintenance. Technical Report TONES-D13, Tones Consortium, January 2007. Available at `http://www.tonesproject.org/`.

[7] F. Baader, D. Calvanese, G. De Giacomo, P. Fillottrani, E. Franconi, B. Cuenca Grau, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, R. Möller, B. Parsia, P. Patel-Schneider, R. Rosati, B. Suntisrivaraporn, and S. Tessaris. Formalisms for representing Ontologies: State of the art survey. Technical Report TONES-D06, Tones Consortium, May 2006. Available at `http://www.tonesproject.org/`.

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook – Theory, Implementaion and Applications*. Cambridge University Press, 2003.

[9] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[10] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, Pittsburgh, PA, USA, 2005.

[11] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms for reasoning about web services. LTCS-Report 05-02, TU Dresden, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[12] 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.

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

[14] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[15] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 270–281. Morgan Kaufmann, 1992.

[16] 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.

[17] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proc. of the 9th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer, 2000.

[18] S. Bechhofer, I. Horrocks, and D. Turi. The OWL instance store: System description. In *Proceedings CADE-20*, LNCS. Springer Verlag, 2005.

[19] Sean Bechhofer, R. Möller, and Peter Crowther. The DIG Description Logic Interface. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7*, 2003.

[20] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66:1–72, 1966.

[21] Tim Berners Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[22] A. Borgida, M. Lenzerini, and R. Rosati. Description logics for databases. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 462–484. Cambridge University Press, 2003.

[23] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[24] Alexander Borgida and Ronald J. Brachman. Conceptual modeling with description logics. In Baader et al. [8], chapter 10, pages 349–372.

[25] 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.

[26] Dan Brickley and R. V. Guha. Resource description framework (rdf) schema specification 1.0. Technical report, World Wide Web Consortium, March 2000.

[27] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. The size of a revised knowledge base. *Artificial Intelligence*, 115(1):25–64, 1999.

[28] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.

[29] D. Calvanese, E. Franconi, B. Glimm, B. Cuenca Grau, I. Horrocks, A. Kaplunova, D. Lembo, M. Lenzerini, C. Lutz, R. Möller, R. Rosati, U. Sattler, S. Tessaris, and A.-Y. Turhan. D10: Tasks for Ontology Access, Processing, and Usage. Project deliverable, TONES, 2006. http://www.tonesproject.org.

[30] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.

[31] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998)*, pages 149–158. ACM Press, 1998.

[32] D. Calvanese, B. Cuenca Grau, E. Franconi, I. Horrocks, A. Kaplunova, C. Lutz, R. Möller, B. Sertkaya, S. Tessaris, and A.-Y. Turhan. D15: Software Tools for Ontology Design and Maintenance. Project deliverable, TONES, 2007. http://www.tonesproject.org.

[33] Diego Calvanese, Bernardo Cuenca Grau, Giuseppe De Giacomo, Enrico Franconi, Ian Horrocks, Alissa Kaplunova, Domenico Lembo, Maurizio Lenzerini, Carsten Lutz, Davide Martinenghi, Ralf Möller, Riccardo Rosati, Sergio Tessaris, and Anni-Yasmin Turhan. Common framework for representing ontologies. Technical Report TONES-D08, TONES Consortium, July 2006. Available at `http://www.tonesproject.org/`.

[34] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[35] 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.

[36] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 2007. To appear.

[37] 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.

[38] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. What can knowledge representation do for semi-structured data? In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 205–210, 1998.

[39] 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.

[40] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, pages 107–118. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-53/`, 2002.

[41] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[42] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Knowledge representation approach to information integration. In *Proc. of AAAI Workshop on AI and Information Integration*, pages 58–65. AAAI Press/The MIT Press, 1998.

[43] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[44] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logics. *ACM Transactions on Computational Logic (TOCL)*, 2007. To appear.

[45] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Nineth ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[46] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, March 1976.

[47] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[48] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, October 1992.

[49] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[50] Reginald Ferber. *Information Retrieval*. dpunkt.verlag, 2003.

[51] Kenneth D. Forbus. Introducing actions into qualitative simulations. In *International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1273–1279. Morgan Kaufman, 1989.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[53] M. Garcia, A. Kaplunova, and R. Möller. Model Generation in Description Logics: What Can We Learn From Software Engineering? Technical report, Institute for Software Systems (STS), Hamburg University of Technology, 2007. to appear.

[54] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Boook*. Prentice Hall, 2092.

[55] B. Glimm, I. Horrocks, and U. Sattler. Conjunctive query answering for description logics with transitive roles. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*, 2006. to appear.

[56] 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.

[57] Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of IJCAI 2007*, pages 399–404, 2007.

[58] Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL repositories. In *Proc. of the Second Int. Semantic Web Conf. (ISWC 2003)*, number 2870 in LNCS, pages 613–627. Springer Verlag, 2003.

[59] Y. Guo, Z. Pan, and J. Heflin. An evaluation of knowledge base systems for large OWL datasets. In *Proc. of the Third Int. Semantic Web Conf. (ISWC 2004)*, LNCS. Springer Verlag, 2004.

[60] V. Haarslev and R. Möller. RACER System Description. In *Int. Joint Conference on Automated Reasoning, IJCAR '01*, 2001.

[61] V. Haarslev and R. Möller. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004, Whistler, BC, Canada, June 2-5*, pages 163–173, 2004.

[62] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science, pages 29–44. Springer, June 2001.

[63] V. Haarslev, R. Möller, and M. Wessel. Description logic inference technology: Lessons learned in the trenches. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proc. International Workshop on Description Logics*, 2005.

[64] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and role-forming predicates. *J. of Logic and Computation*, 9(3):351–384, 1999.

[65] Volker Haarslev and Ralf Möller. Consistency testing: The RACE experience. In *Proceedings International Conference Tableaux'2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 57–61. Springer, 2000.

[66] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 161–166. Morgan-Kaufmann, 2001.

[67] Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.

[68] Volker Haarslev, Ralf Möller, and Michael Wessel. Querying the semantic web with racer + nrql. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, 2004.

[69] Volker Haarslev, Ralf Möller, and Michael Wessel. A high performance semantic web query answering engine. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, 2005.

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

[71] A. Herzig. The PMA revisited. In *KR'96*. Morgan Kaufmann, 1996.

[72] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. Technical report, University of Manchester, 2006.

[73] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. Technical report, World Wide Web Consortium, May 2004.

[74] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, 1999.

[75] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Computer Science, Germany, 2000. Springer Verlag.

[76] I. Horrocks and S. 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.

[77] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000), Breckenridge, Colorado, USA, April 11-15, 2000*, pages 285–296, April 2000.

[78] Ian Horrocks. Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science, 1997.

[79] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR98)*, pages 636–647, 1998.

[80] Ian Horrocks, Deborah L. McGuinness, and Christopher A. Welty. Digital libraries and web-based information systems. pages 427–449, 2003.

[81] Ian Horrocks and Peter Patel-Schneider. The generation of DAML+OIL. In Carole Goble, Deborah L. McGuinness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Proceedings of the International Workshop in Description Logics 2001 (DL2001)*, number 49 in CEUR-WS (http://ceur-ws.org/), pages 30–35, 2001.

[82] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[83] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1), 2003.

[84] 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.

[85] Ian Horrocks and Sergio Tessaris. Querying the Semantic Web: a formal approach. In *Proc. of the 1st Int. Semantic Web Conf. (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2002.

[86] U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, pages 466–471, 2005.

[87] Ulrich Hustadt, Boris Motik, and Ulrike Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *Proc. of the 11th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, pages 21–35, 2004.

[88] Ulrich 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.

[89] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[90] E. Karabaev and C. Lutz. Mona as a dl reasoner. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, CEUR-WS, 2004.

[91] A. Kaya, S. Melzer, R. Möller, S. Espinosa, and M. Wessel. Towards a Foundation for Knowledge Management: Multimedia Interpretation as Abduction. In *Proc. of the Int. Workshop on Description Logics, DL '07*, 2007.

[92] Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logics*, 69(2–3):243–268, 1994.

[93] Natasha Kurtonina and Maarten de Rijke. Expressiveness of concept expressions in first-order description logics. *Artificial Intelligence*, 107(2):303–333, 1999.

[94] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-Connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.

[95] H. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 3(18), 1988.

[96] Hector J. Levesque, Raymond Reiter, Lesperance Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: a logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[97] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining horn rules and description logics. In *European Conference on Artificial Intelligence*, pages 323–327, 1996.

[98] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence Journal*, 46:365–376, 1990.

[99] Fangzhen Lin. Embracing causality in specifying the indeterminate effects of actions. In B. Clancey and D. Weld, editors, *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-96)*, pages 670–676, Portland, OR, August 1996. MIT Press.

[100] H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic ABoxes. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 46–56. AAAI Press, 2006.

[101] Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating ABoxes. LTCS-Report 05-10, Technical University Dresden, 2005. see http://lat.inf.tu-dresden.de/research/reports.html.

[102] C. Lutz and M. Milicic. A tableau algorithm for dls with concrete domains and gcis. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, number 147 in CEUR-WS, 2005.

[103] C. Lutz, U. Sattler, and F. Wolter. Modal logics and the two-variable fragment. In *Annual Conference of the European Association for Computer Science Logic CSL'01*, LNCS, Paris, France, 2001. Springer Verlag.

[104] C. Lutz and F. Wolter. Modal logics of topological relations. In *Proceedings of Advances in Modal Logics 2004*, 2004.

[105] Carsten Lutz. Inverse roles make conjunctive queries hard. In *Proc. of DL 2007*, 2007.

[106] Frank Manola and Eric Miller. Rdf primer. Technical report, World Wide Web Consortium, February 2004.

[107] B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

[108] S. McIlraith, T.C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems. Special Issue on the Semantic Web*, 16(2):46–53, 2001.

[109] Sheila A. McIlraith and Tran Cao Son. Adapting GOLOG for composition of semantic web services. In Dieter Fensel, Fausto Giunchiglia, Deborah McGuinness, and Mary-Anne Williams, editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

[110] R. Möller, K. Hidde, R. Joswig, Th. Mantay, and B. Neumann. Bericht über das projekt band: Benutzeradaptiver netzinformationsdienst. Technical Report LKI-Memo LKI-M-99/01, Labor für Künstliche Intelligenz, Fachbereich Informatik, Universität Hamburg, 1999.

[111] R. Möller and M. Wessel. Terminological default reasoning about spatial information: A first step. In *Proc. of COSIT'99, International Conference on Spatial Information Theory, Stade*, pages 189–204. Springer-Verlag, 1999.

[112] 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.

[113] B. Motik, R. Volz, and A. Maedche. Optimizing query answering in description logics using disjunctive deductive databases. In *Proceedings of the 10th International Workshop on Knowledge Representation Meets Databases (KRDB-2003)*, pages 39–50, 2003.

[114] M.P.Shanahan. *Solving the Frame Problem*. MIT Press, 1997.

[115] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[116] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann, 1991.

[117] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.

[118] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[119] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of AAAI 2006*, 2006. To appear.

[120] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. In *Proc. of DL 2006*, 2006.

[121] Leszek Pacholski, Wiesław Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM Journal on Computing*, 29(4):1083–1117, August 2000.

[122] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[123] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46:325 – 361, 1999.

[124] I. Pratt-Hartmann. Counting quantifiers and the stellar fragment. Available at The Mathematics Preprint Server, http://www.mathpreprints.com, 2003.

[125] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language, and Information*, 14(3):369–395, 2005.

[126] D. A. Randell, Z. Cui, and A. G. Cohn. A Spatial Logic based on Regions and Connections. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning*, pages 165–176, 1992.

[127] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[128] R. Reiter. *Knowledge in Action*. MIT Press, 2001.

[129] J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1–2):69–123, 1999.

[130] J. Renz and B. Nebel. Efficient Methods for Qualitative Spatial Reasoning. *Journal of Artificial Intelligence Research*, 15:289–318, 2001.

[131] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006)*, pages 356–365, 2006.

[132] E. Sandewall. *Features and Fluents*. Oxford University Press, 1994.

[133] Richard Scherl and Hector Levesque. Knowledge, action, and the frame problem. *AIJ*, 144(1):1–39, 2003.

[134] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[135] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[136] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[137] Guy L. Steele, Jr. *Common Lisp – The Language, Second Edition*. Digital Press, 1990.

[138] The OWL-S Coalition. OWL-S 1.1 (beta) draft release, 2004. http://www.daml.org/services/owl-s/1.1/.

[139] The W3C Consortium. The web ontology language (OWL), 2005. http://www.w3.org/2004/OWL/.

[140] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998.

[141] Michael Thielscher. Nondeterministic actions in the fluent calculus: Disjunctive state update axioms. In S. Hölldobler, editor, *Intellectics and Computational Logic*, pages 327–345. Kluwer Academic, 2000.

[142] Michael Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *KR*, pages 109–120, Breckenridge, CO, April 2000. Morgan Kaufmann.

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

[144] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, 2000.

[145] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithöner. DIG 2.0 Towards a Flexible Interface for Description Logic Reasoners. In B. Cuenca Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *In Proceedings of the second international workshop OWL: Experiences and Directions*, November 2006.

[146] Peter van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

[147] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, http://www.w3.org/tr/owl-guide/, 2003.

[148] 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.

[149] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[150] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, U. Mehlig, Th. Neumann, M. Völter, and U. Zdun. *Software-Architektur – Grundlagen – Konzepte – Praxis*. Elsevier Spektrum Akademischer Verlag, 1 edition, 2005.

[151] H. Wache, T. Vögele, T. Visser, U. Stuckenschmidt, H. Schuster, G. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches, 2001.

[152] Timo Weithöner, Thorsten Liebig, and Günther Specht. Storing and Querying Ontologies in Logic Databases. In *Proceedings of The first International Workshop on Semantic Web and Databases (SWDB'03)*, pages 329 – 348, Berlin, Germay, September 2003.

[153] M. Wessel. On Spatial Reasoning with Description Logics - Position Paper. In I. Horrocks and S. Tessaris, editors, *Proceedings of the International Workshop on Description Logics 2002 (DL2002)*, number 53 in CEUR-WS, pages 156–163, Toulouse, France, April 19–21 2002. RWTH Aachen.

[154] M. Wessel. Qualitative Spatial Reasoning with the $\mathcal{ALCI}_{\mathcal{RCC}}$-family – First Results and Unanswered Questions. Technical Report FBI–HH–M–324/03, University of Hamburg, Computer Science Department, May 2003.

[155] M. Wessel and R. Möller. A high performance semantic web query answering engine. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.

[156] M. Wessel and R. Möller. A Flexible DL-based Architecture for Deductive Information Systems. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Proc. IJCAR-06 Workshop on Empirically Successful Computerized Reasoning (ESCoR)*, pages 92–111, 2006.

[157] Michael Wessel. Some Practical Issues in Building a Hybrid Deductive Geographic Information System with a DL Component. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases, KRDB '03*.

[158] M. Winslett. Updating logical databases. In D. Gabbay, A. Galton, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 4 "Epistemic and Temporal Reasoning"*, pages 133–174. Oxford University Press, 1995.

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

[160] Marianne Winslett. *Updating Logical Databases*. Cambridge University Press, Cambridge, England, 1990.

[161] Y. Zhang and N. Foo. Updating knowledge bases with disjunctive information. *Computational Intelligence*, 16:1–22, 2000.

[162] Z. Zhang. Ontology query languages for the semantic web: A performance evaluation. Master's thesis, University of Georgia, 2005.