Ontology Query Answering on Databases

Jing Mei[1,2], Li Ma[2], Yue Pan[2]

[1] Department of Information Science
Peking University
Beijing 100871, China
{mayyam} @ is.pku.edu.cn
[2] IBM China Research Lab
Beijing 100094, China
{malli, panyue} @ cn.ibm.com

Abstract. With the fast development of Semantic Web, more and more RDF and OWL ontologies are created and shared. The effective management, such as storage, inference and query, of these ontologies on databases gains increasing attention. This paper addresses ontology query answering on databases by means of Datalog programs. Via epistemic operators, integrity constraints are introduced, and used for conveying semantic aspects of OWL that are not covered by Datalog-style rule languages. We believe such a processing suitable to capture ontologies in the database flavor, while keeping reasoning tractable. Here, we present a logically equivalent knowledge base whose (sound and complete) inference system appears as a Datalog program. As such, SPARQL query answering on OWL ontologies could be solved in databases. Bi-directional strategies, taking advantage of both forward and backward chaining, are then studied to support this kind of customized Datalog programs, returning exactly answers to the query within our logical framework.

1 Introduction

The Resource Description Framework (RDF) [25] has been recognized as a popular way to represent information in the Semantic Web, accompanying with a standardized query language, SPARQL [27]. An important vocabulary extension of RDF is the Web Ontology Language (OWL) [24], whose formalisms rely closely on the Description Logic (DL) [2]. The Semantic Web Rule Language (SWRL) [28] arises, when rules are considered to work with OWL in a syntactically and semantically coherent manner.

Given an RDF document, having information of ontologies (i.e. upgraded to an OWL document) even rules (i.e. further upgraded to a SWRL document), a SPARQL query is to extract implicit and explicit RDF data, with query answering as an underlying reasoning service. Compared with weak DL query languages, SPARQL is expressible for the union of conjunctive queries on RDF triples, not only concerning traditional DL queries (e.g. instantiation, realization and retrieval [14]), but also allowing predicates (could be DL classes and properties) being queried as variables.

Database (DB) technologies provide a solid support for various data-intensive applications and could be used for ontology management. So, SPARQL queries on ontologies possibly work in a similar way as SQL queries on relational data in databases. Meanwhile, the intention of making the Semantic Web more acceptable to the industry is another underlying promotion to impede the connection to database communities [9]. However, relational DBs are not rich enough to capture semantics implied in ontologies. Thus, Datalog is regarded as a suitable intermedium, and the next section motivates this opinion in detail.

Technically, this paper generalizes a (negation-free) Datalog program P w.r.t. a given DL KB $\Sigma =< \mathcal{T}, \mathcal{A} >$, whose EDB (extensional database) consists of the inferred DL TBox $\mathcal{T}^*$ and the original DL ABox $\mathcal{A}$, and whose IDB (intensional database) is composed of 7 inference rules devised by DL semantics. We state that P is a syntax variant of an inference system Γ, and Γ is proved sound and complete with a specified MKNF-DL [10] KB Σ', where Σ', assuming satisfiable, extends Σ with *integrity constraints* (ICs). By *epistemic interpretations*, those ICs exactly reflect the semantic discrepancy between DL and DB, making query answering in DL accessible to the DB community. As such, we elaborate a strategy, which benefits from the tractable data complexity of Datalog, to address popular SPARQL queries on databases, provided by RDF data obtained from OWL documents. Within such a framework, it is possible to futher support SWRL rules which are user-defined Datalog rules sharing predicates with the ontology classes and properties.

The paper is organized as follows. Section 2 is our motivation including an analysis to encountered problems and existing approaches. Section 3 introduces the preliminaries and notions used in this paper. In Section 4, a so-called DL2DB KB Σ' w.r.t. a given DL KB Σ is defined, along with a natural deduction system Γ consisting of DL-driven inference rules. Also, the correspondence of Σ' to its original Σ, as well as the soundness and completeness of Γ with Σ', are presented. A Datalog program, corresponding to Γ, is demonstrated in Section 5 for query answering in Σ. There, we discuss computation strategies and evaluation approaches as well. Section 6 envisions the desirable extension with user-defined rules, and shows preliminary experiments in our prototype implementation. Finally, the conclusion is drawn in Section 7. (Readers can find more details on the proofs in the paper at `http://www.is.pku.edu.cn/~mayyam/appendix.pdf`.)

2 Motivation

Our objective is to perform SPARQL query answering on databases which are used to store RDF data (including OWL ontologies). However, it has been studied in DL-Lite [6] that data complexity of query answering in DL has a LogSpace boundary, above which query answering is not expressible as a first-order logic formula and hence a SQL query. However, most DLs are more expressive than DL-Lite. The worst case is an unrestricted combination of OWL and rules, such as SWRL, leading to the undecidability of interesting reasoning problems [28]. An ExpTime complexity is reobtained, for query answering in a less expressive

extension to OWL with DL-safe rules [19], but the exponential data complexity greatly weakens the efficiency.

The discrepancy between DL and DB is actually remarkable. As well-known, DL is based on an open world assumption (OWA) permitting incomplete information in an ABox, while DB adopts a closed world assumption (CWA) requiring information always understood as complete. The unique name assumption (UNA) is often emphasized in DB but not in DL. OWL Flight [9], furthermore, clarifies restrictions in DL and constraints in DB, of which the former is to infer and the latter to check. When negation comes, DBs prefer to "non-monotonic negation", while DLs rely on "monotonic negation".

Facing to these open issues, various proposals have been presented. According to the engines which do reasoning indeed, existing ontology persistent systems can be roughly divided into two categories: DL-based and rule-based. The Instance Store [4] is a representative of DL-based systems, where DB serves mainly for voluminous storage and convenient retrieval, and classical DL tableaux algorithms help make implicit information explicit. In the Instance Store, whatever queries are (particularly, those queries involving properties and variables can be handled by rolling-up techniques [14]), a reduction to checking the KB unsatisfiability provides support for query answering. Rule-based approaches are a bit different, which intuitively translate the meaning of DL constructors into rules. Due to the expressive power of rules, those DL constructors (e.g. existential restrictions) are either partially forbidden (as DLP [12] does) or assigned new meanings (as OWL Flight [9] does). Unlike DL tableaux algorithms, the evaluation of queries adopts strategies by forward chaining or backward chaining. More tractably, DL-lite [6] is proposed to execute the ABox using a SQL engine, whose language itself is restrictive while keeping low complexity of reasoning, namely polynomial in the size of instances in the knowledge base.

Since rule-based approaches are more extensible to SWRL, we make an attempt to give inference rules as a translation of DL semantics, while RDF data (from OWL documents) is stored in databases. Concentrating on query answering, this paper introduces a DL2DB deduction system Γ and studies which sublanguage of DLs is equivalent to the so-called DL2DB. Surely, DLP (Description Logic Programs [12]) is a nice measure, as cited by most related work. However, inspired by MKNF-DL (Description Logics of Minimal Knowledge and Negation as Failure [10]), also motivated by bridging DL and DB, we exploit integrity constraints to a DL KB Σ, resulting in a special MKNF-DL KB Σ'. Checking satisfiability of (DB) integrity constraints has been well investigated (e.g. in [5]), and our proposed ICs, in the DL setting, admit of $C \sqcup D$ (respectively, $\exists P.C$) being *known* only if holding an autoepistemic belief of either C or D (respectively, P and C). That is, we contribute Σ' as a logically equivalent version of Σ in the sense of query answering. Particularly, the DL2DB system Γ proves sound and complete with Σ' for non-epistemic queries. With a focus on query answering, we assume both Σ and Σ' are satisfiable. Generally speaking, it makes little sense, in classical logics, when everything is possible to be entailed by inconsis-

tent KBs. Also, checking satisfiability of a DL KB or a MKNF-DL KB has been studied in [2] and [10], but not scoped in this paper.

We remark this paper does not propose to "change" or "weaken" the semantics of a DL KB. Instead, for keeping DL in classical first-order semantics, we move to a MKNF-DL "world" whose unique epistemic model is identical to that of DL, provided by integrity constraints on demand. In other words, we capture an epistemic perception for those rule-based approaches, and gain an insight into integrity constraints for DL constructors, rather than tying to change or weaken the classical DL semantics.

On the other hand, we do realize nonmonotonic features are gaining increasing interest in the context of the Semantic Web initiatives [26], and the paper [8] provides a good survey. A latest work is [18], which proposes hybrid MKNF KBs integrating decidable DLs with nonmonotonic rules. However, our paper here belongs to the direction of research for query answering over ontologies relying over database technologies by making use, in a "natural" way as a rational agent does, of well established formalizations and computing mechanisms.

3 Preliminaries

Consider the main layers of the DL family bottom-up [2][13], $\mathcal{ALC}$ is a basic and simple language, permitting class descriptions via $C \sqcap D, C \sqcup D, \neg C, \forall P.C$, and $\exists P.C$, where C, D are classes and P is a property. Augmented by transitive properties, $\mathcal{ALC}$ becomes $\mathcal{ALC}_{R^+}$, denoted by $\mathcal{S}$ in the following. $\mathcal{SI}$ is an extension to $\mathcal{S}$ with inverse properties, followed by $\mathcal{SHI}$ with property hierarchies. It is called $\mathcal{SHIF}$ if extending by functional restrictions, $\mathcal{SHIN}$ if by cardinality restrictions, and $\mathcal{SHIQ}$ if by qualified number restrictions. Support for datatype predicates (e.g. string, integer) brings up the concrete domain of $\mathbf{D}$, and using nominals $\mathcal{O}$ helps construct classes with singleton sets.

With the expected pervasive use of OWL, $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ are paid more attention: one is the syntax variant to OWL Lite and the other is to OWL DL. This paper currently takes $\mathcal{SHI}$ into account, and more expressive extensions will be explored in our ongoing work. Alternately, DB built-in features, such as arithmetic operators and aggregate functions, might be considered as a workaround for $\mathcal{F}, \mathcal{N}, \mathcal{Q}$ and $\mathbf{D}$, while list operations for $\mathcal{O}$. In the following, if not stated otherwise, C, D denote $\mathcal{SHI}$ classes and P, Q denote $\mathcal{SHI}$ properties.

A DL KB Σ is defined as a pair $\Sigma =< \mathcal{T}, \mathcal{A} >$. The TBox $\mathcal{T}$ is a finite set of class and property subsumptions having the form of $C \sqsubseteq D$ and $P \sqsubseteq Q$, resp. The ABox $\mathcal{A}$ is a finite set of class and property assertions having the form of $C(a)$ and $P(a, b)$. Also, an *interpretation* $\mathcal{I} = (\Delta, \bullet^{\mathcal{I}})$ consists of a nonempty set Δ (the domain of $\mathcal{I}$) and a function $\bullet^{\mathcal{I}}$ (the interpretation function of $\mathcal{I}$) that maps every class to a subset of Δ and every property to a subset of $\Delta \times \Delta$. An interpretation $\mathcal{I}$ is a model of a DL KB Σ (denoted as $\mathcal{I} \models \Sigma$) iff every sentence (subsumption or assertion) of Σ is satisfied in $\mathcal{I}$. For a complete presentation of other definitions, such as the satisfiability of sentences, we direct readers to the classical DL handbook [2]. Query answering for $q(\bar{x})$ in Σ attempts to receive

all ground substitutions $\bar{t}$ to $\bar{x}$ such that $\Sigma \models q(\bar{t})$, and two KBs Σ and Σ' are equivalent in the sense of query answering iff those obtained results are identical in both Σ and Σ' [6].

Next, a more sophisticated language, namely MKNF-DL [10], is sketched. One epistemic operator $\mathbf{K}$ works for minimal knowledge, and the other epistemic operator $\mathbf{A}$ plays a role to default assumption. The syntax of MKNF-DL extends DL with $\mathbf{K}C, \mathbf{K}P, \mathbf{A}C$ and $\mathbf{A}P$, where C is a DL class and P is a DL property. The semantics of MKNF-DL resorts to *epistemic interpretations*. An epistemic interpretation is a triple $(\mathcal{I}, \mathcal{M}, \mathcal{N})$ where $\mathcal{I}$ is a (first-order) interpretation and $\mathcal{M}, \mathcal{N}$ are sets of (first-order) interpretations. Non-epistemic classes and properties are interpreted same as in $\mathcal{I}$, i.e. $C^{\mathcal{I},\mathcal{M},\mathcal{N}} = C^{\mathcal{I}}$ and $P^{\mathcal{I},\mathcal{M},\mathcal{N}} = P^{\mathcal{I}}$. The other semantic conditions state that:

$\top^{\mathcal{I},\mathcal{M},\mathcal{N}} = \Delta$; $\bot^{\mathcal{I},\mathcal{M},\mathcal{N}} = \emptyset$; $(\neg C)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \Delta \backslash C^{\mathcal{I},\mathcal{M},\mathcal{N}}$;
$(C \sqcap D)^{\mathcal{I},\mathcal{M},\mathcal{N}} = C^{\mathcal{I},\mathcal{M},\mathcal{N}} \cap D^{\mathcal{I},\mathcal{M},\mathcal{N}}$; $(C \sqcup D)^{\mathcal{I},\mathcal{M},\mathcal{N}} = C^{\mathcal{I},\mathcal{M},\mathcal{N}} \cup D^{\mathcal{I},\mathcal{M},\mathcal{N}}$;
$(\exists P.C)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \{a \in \Delta | \exists b.(a,b) \in P^{\mathcal{I},\mathcal{M},\mathcal{N}}$ and $b \in C^{\mathcal{I},\mathcal{M},\mathcal{N}}\}$;
$(\forall P.C)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \{a \in \Delta | \forall b.(a,b) \in P^{\mathcal{I},\mathcal{M},\mathcal{N}}$ implies $b \in C^{\mathcal{I},\mathcal{M},\mathcal{N}}\}$;
$(\mathbf{K}C)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \bigcap_{\mathcal{J} \in \mathcal{M}} (C^{\mathcal{J},\mathcal{M},\mathcal{N}})$; $(\mathbf{K}P)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \bigcap_{\mathcal{J} \in \mathcal{M}} (P^{\mathcal{J},\mathcal{M},\mathcal{N}})$;
$(\mathbf{A}C)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \bigcap_{\mathcal{J} \in \mathcal{N}} (C^{\mathcal{J},\mathcal{M},\mathcal{N}})$; $(\mathbf{A}P)^{\mathcal{I},\mathcal{M},\mathcal{N}} = \bigcap_{\mathcal{J} \in \mathcal{N}} (P^{\mathcal{J},\mathcal{M},\mathcal{N}})$;
For any property Q being the inverse of P, if $(a,b) \in P^{\mathcal{I},\mathcal{M},\mathcal{N}}$ then $(b,a) \in Q^{\mathcal{I},\mathcal{M},\mathcal{N}}$;
For any transitive property P, if $(a,b),(b,c) \in P^{\mathcal{I},\mathcal{M},\mathcal{N}}$ then $(a,c) \in P^{\mathcal{I},\mathcal{M},\mathcal{N}}$.

A set of interpretations $\mathcal{M}$ is an epistemic model for Σ (denoted as $\mathcal{M} \models \Sigma$) iff the structure $(\mathcal{M}, \mathcal{M})$ satisfies Σ and, for each set of interpretations $\mathcal{M}'$, if $\mathcal{M} \subset \mathcal{M}'$ then $(\mathcal{M}', \mathcal{M})$ does not satisfy Σ. A structure $(\mathcal{M}, \mathcal{N})$ satisfies Σ (denoted as $(\mathcal{M}, \mathcal{N}) \models \Sigma$) iff each interpretation $\mathcal{I} \in \mathcal{M}$ is such that every sentence (subsumption or assertion) of Σ is satisfied in the epistemic interpretation $(\mathcal{I}, \mathcal{M}, \mathcal{N})$. The paper of MKNF-DL [10] provides a complete presentation of other definitions, such as the satisfiability of sentences, and we follow its convention. Notice that, for any (non-epistemic) DL KB Σ, it has one and only one epistemic model, i.e. the set of all first-order models for Σ, denoted as $\mathcal{M}(\Sigma)$.

Finally, this paper adopts assumptions that are suitable for the semantics of MKNF (cf. [8][10][18]): (1) Every first-order interpretation is over the same fixed, countably infinite domain; (2) There is a one-to-one correspondence between individuals in the language and elements in the domain. Thus, the set of all individuals $\mathcal{O}$ is fixed to $\mathcal{I}$, i.e., $\Delta = \mathcal{O}$, and we denote the interpretation of $a \in \mathcal{O}$ simply as a itself, i.e., $a^{\mathcal{I}} = a$. The assumption of (2) also implies that two distinct individuals denote two distinct elements, referred as to UNA.

4 Bridging DL and DB

Given a DL KB Σ, a specified MKNF-DL KB, namely the DL2DB KB Σ', is studied. We guarantee the equivalency of Σ' and Σ in the sense of query answering, for non-epistemic queries. And then, we generalize an inference system, namely the DL2DB system Γ, and this contribution of Γ is posed by its soundness and completeness with Σ', building a bridge between DL and DB.

4.1 The DL2DB KB

Inspired by representing integrity constraints (ICs) in MKNF-DL [10], we propose ICs for some DL constructors to strengthen beliefs, meeting the requirement that w.r.t. certain information the KB is prone to be complete.

Before introducing definitions, we recall how to formalize ICs using an example from [10]. For instance, a TBox $\mathcal{T}$: $\mathbf{K}\texttt{employee} \sqsubseteq \mathbf{A}\texttt{male} \sqcup \mathbf{A}\texttt{female}$ corresponds to an IC that "each known employee must be known to be either male or female". Having only one assertion in ABox $\mathcal{A}$: `employee(Bob)`, makes this KB $< \mathcal{T}, \mathcal{A} >$ lack of epistemic models. That is, this IC is violated. Turning to a system level, we require each known DL class of $C \sqcup D$ must be known to be either C or D, viz. `sub1` as defined below, while `sub2` is for $\exists P.C$. Considering the discrepancy between DL and DB, we believe that ICs bridge them in a semantic "pay-as-you-go" manner.

We use $\texttt{clos}(C)$ for the closure of a class C, and $\texttt{clos}(C)$ is the smallest set containing C that is closed under subclasses and negation (in Negation Normal Form), while the size of $\texttt{clos}(C)$ is linear in the length of C [13]. Given a set of classes M, $\texttt{clos}(M) = \bigcup_{C \in M} \texttt{clos}(C)$, the size of $\texttt{clos}(M)$ is polynomial in the size of M.

Definition 1. *Let $\Sigma =< \mathcal{T}, \mathcal{A} >$ be a DL KB, and Σ_C be the set of classes occurring in Σ. A DL2DB KB w.r.t. Σ is $\Sigma' =< \mathcal{T}', \mathcal{A} >$ where $\mathcal{T}' = \mathcal{T} \cup$ `sub1` $\cup$ `sub2`,*
$\texttt{sub1}=\{\mathbf{K}(C \sqcup D) \sqsubseteq \mathbf{A}C \sqcup \mathbf{A}D \mid C \sqcup D \in \texttt{clos}(\Sigma_C)\}$ *and*
$\texttt{sub2}=\{\mathbf{K}(\exists P.C) \sqsubseteq \exists \mathbf{A}P.\mathbf{A}C \mid \exists P.C \in \texttt{clos}(\Sigma_C)\}$.

The following proposition helps gain insights into the nature of epistemic models for Σ' and Σ. Due to space limitation, the detailed proof can be found at `http://www.is.pku.edu.cn/~mayyam/appendix.pdf`.

Proposition 1. *Let Σ be a DL KB, and Σ' be a DL2DB KB w.r.t. Σ. A set of interpretations $\mathcal{M}$ is an epistemic model for Σ' iff (1) $\mathcal{M}$ is an epistemic model for Σ; (2) for each subsumption $\varphi \in$ `sub1`$\cup$`sub2` in Σ', $(\mathcal{M}, \mathcal{M})$ satisfies φ.*

As pointed out above, the (non-epistemic) DL KB Σ has a unique epistemic model, namely $\mathcal{M}(\Sigma)$, consisting of all first-order models for Σ. This proposition indicates that Σ' has the same unique epistemic model as $\mathcal{M}(\Sigma)$ if the structure $(\mathcal{M}(\Sigma), \mathcal{M}(\Sigma))$ satisfies all subsumptions in `sub1`$\cup$`sub2`, otherwise Σ' is unsatisfiable. Thus, under the assumption that both Σ and Σ' are satisfiable, the two KBs Σ and Σ' are equivalent in the sense of query answering.

It possibly happens that Σ is satisfiable but Σ' is not, which implies some of those ICs have been violated. In this case, we fail to returning complete answers to queries via Σ', although incomplete information in Σ would not attack Σ itself. For example, a DL KB Σ, having a TBox $\mathcal{T}$: $\texttt{male} \sqsubseteq \texttt{person}$ and $\texttt{female} \sqsubseteq \texttt{person}$ in addition to an ABox $\mathcal{A}$: $\texttt{male} \sqcup \texttt{female}(\texttt{Bob})$, is satisfiable and entails `person(Bob)` but neither `male(Bob)` nor `female(Bob)`. Obviously, the IC of $\mathbf{K}(\texttt{male} \sqcup \texttt{female}) \sqsubseteq \mathbf{A}\texttt{male} \sqcup \mathbf{A}\texttt{female}$ is violated, making this Σ' unsatisfiable. In real DB-based applications, it is highly possible to have complete information on `Bob`'s gender, which leads to `person(Bob)` in Σ' and reobtaining the satisfiability of Σ'.

4.2 The DL2DB System

DL TBox reasoning has been well-developed, but scalable DL ABox reasoning deserves more investigation. Instead of using classical DL tableaux calculus, we aim at exploiting an alternative way to ABox reasoning on databases, which are initialized by an inferred TBox and an original ABox.

Since the complexity of deciding $\mathcal{SHI}$ DL class satisfiability is ExpTime-complete [13], computing the closure of the TBox is in coExpTime, provided that C is subsumed by D iff $C \sqcap \neg D$ is unsatisfiable [2]. Although, such computational impact might be non-negligible, TBoxes are relatively fixed and certain data preprocessing at the back end is feasible for applications, to some extent.

So far, a DL2DB system Γ w.r.t. a DL KB $\Sigma =< \mathcal{T}, \mathcal{A} >$ is constructed as below. Starting by initialization, a TBox taxonomy $\mathcal{T}^* = \{C \sqsubseteq D | \Sigma \models (C \sqsubseteq D)\} \cup \{P \sqsubseteq Q | \Sigma \models (P \sqsubseteq Q)\}$, derived from external DL reasoners, is uploaded to Γ together with the original ABox $\mathcal{A}$.

Initialization:

TBox: If $\varphi \in \mathcal{T}^*$, then $\varphi \in \Gamma$.
ABox: If $\varphi \in \mathcal{A}$, then $\varphi \in \Gamma$.

Inference rules:

$\in$: If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.
$\sqcap$: If $\Gamma \vdash C(a)$ and $\Gamma \vdash D(a)$, then $\Gamma \vdash (C \sqcap D)(a)$.
$\exists$: If $\Gamma \vdash P(a, b)$ and $\Gamma \vdash C(b)$, then $\Gamma \vdash \exists P.C(a)$.
$\forall$: If $\Gamma \vdash P(a, b)$ and $\Gamma \vdash \forall P.C(a)$, then $\Gamma \vdash C(b)$.
$\sqsubseteq_T$: If $\Gamma \vdash (C \sqsubseteq D)$ and $\Gamma \vdash C(a)$, then $\Gamma \vdash D(a)$.
$\sqsubseteq_P$: If $\Gamma \vdash (P \sqsubseteq Q)$ and $\Gamma \vdash P(a, b)$, then $\Gamma \vdash Q(a, b)$.
P_i: If $\Gamma \vdash P(a, b)$, then $\Gamma \vdash Q(b, a)$, where Q is inverse of P.
P_t: If $\Gamma \vdash P(a, b)$ and $\Gamma \vdash P(b, c)$, then $\Gamma \vdash P(a, c)$, where P is transitive.

Symbols of a, b, C, D, P, Q etc. will be instantiated by corresponding individuals, classes and properties in Σ. Classical $\mathcal{SHI}$ DL [13] depicts another rule of $\forall_+$: If $\Gamma \vdash P(a, b)$ and $\Gamma \vdash \forall Q.C(a)$, where P is transitive and $\Gamma \vdash (P \sqsubseteq Q)$, then $\Gamma \vdash \forall P.C(b)$. However, support for TBox reasoning gives $\forall Q.C \sqsubseteq \forall P.(\forall P.C)$, because of $P \sqsubseteq Q$ where P is transitive. As a result, Rule[$\forall$] covers the situation of Rule[$\forall_+$], i.e., if $\Gamma \vdash P(a, b)$ and $\Gamma \vdash \forall P.(\forall P.C)(a)$, then $\Gamma \vdash \forall P.C(b)$.

Applying these inference rules, a sentence φ is called derivable if $\Gamma \vdash \varphi$, and its derivation length n counts in the times of applying inference rules. A conflict is defined in Γ by the facts that $\Gamma \vdash C(a)$ and $\Gamma \vdash \neg C(a)$. In this paper, conflict-free systems are focused, unless otherwise noted.

As far as we know, these above inference rules, more or less, play a role in most "state of the art" Semantic Web reasoning engines, in particular for those which adopt rule-based approaches. Thus, a corresponding KB fit in with this Γ is expected, and Σ' defined previously happens to be the candidate.

Theorem 1. *Let Σ be a DL KB, Σ' be a satisfiable DL2DB KB w.r.t. Σ, Γ be a conflict-free DL2DB system w.r.t. Σ, and φ be a non-epistemic sentence.*
$\Gamma \vdash \varphi$ if and only if $\Sigma' \models \varphi$.

For proofs of this theorem (or called as soundness and completeness), please refer to the report at `http://www.is.pku.edu.cn/~mayyam/appendix.pdf`.

5 Query Answering

Moving the proposed DL2DB system into practice, a (negation-free) Datalog program is presented in this section. Also, we discuss rewriting techniques which make SPARQL queries processable, even those syntactic sugars involving predicates as variables. By implementing a bi-directional strategy of top-down and bottom-up, we believe that, following some optimization techniques, e.g. Magic Set [3] and Tabling [1], scalable ontology query answering is hopeful.

5.1 SPARQL Queries

SPARQL is a query language for obtaining information from RDF graphs. An RDF graph is a set of triples and each triple consists of a `subject`, a `predicate` and an `object`. From DL perspective, a reserved `predicate` rdf:type, for example, indicates DL class assertions with individuals as the `subject` and DL classes as the `object`. Possibly, a SPARQL query concerns the retrieval of those `objects` standing for DL classes, e.g. asking for all types of a specific individual.

Ignoring non-logical constitutions in SPARQL, such as filters, prefixes and so on, we denote a SPARQL query $q(\bar{x})$ by an expression of the form $\{\bar{x}|\texttt{dnf}(\bar{x},\bar{y})\}$. Here, $\bar{x}$ are the so-called distinguished variables that will be bound with individuals in the KB, and $\bar{y}$ are the non-distinguished variables which are existentially qualified variables [6]. $\texttt{dnf}(\bar{x},\bar{y})$ is a disjunctive normal form of $\texttt{Rel}(sub, pre, obj)$, and `Rel`, being a logical predicate, has three parameters: sub, pre, obj, each of which is either a constant in the RDF DB or a variable in $\bar{x}$ or $\bar{y}$.

A question naturally arises, facing to "constants in the RDF DB": what the RDF DB is and what the constants are. Rewriting techniques are introduced to address the problem. Given a DL KB, $C(a)$ and $P(a,b)$ in the ABox are rewritten by $\texttt{Rel}(a$, rdf:type, $C)$ and $\texttt{Rel}(a, P, b)$, while $C \sqsubseteq D$ and $P \sqsubseteq Q$ in the TBox are rewritten by $\texttt{Rel}(C$, rdfs:subClassOf, $D)$ and $\texttt{Rel}(P$, rdfs:subPropertyOf, $Q)$, where rdf:type, rdfs:subClassOf, and rdfs:subPropertyOf are reserved `predicates`. Actually, rewriting not only provides support for SPARQL queries, but also bridges DL and relational databases. DL constructive classes such as $\exists P.C$ are not straightforwardly expressible inside of DBs. So, we use unique IDs to make them recognizable as for participating in sub, pre, obj. Thus, an RDF DB is a storage of triples w.r.t. a DL KB, where constants are those IDs representing individuals, DL classes and DL properties.

5.2 A Datalog Program

Rewriting techniques, additionally, give our inference rules a new version. For instance, Rule[$\exists$] is depicted as $\texttt{Rel}(a$, rdf:type, $\exists P.C) \leftarrow \texttt{Rel}(a, P, b), \texttt{Rel}(b$, rdf:type, $C)$. Attention should be paid for variable bindings of a, b, P, C, and $\exists P.C$, which impose semantic conditions into the rule body such that a, b are individuals and $\exists P.C$ is a pending DL class together with its affiliated DL property P and class C. Other rules are processed similarly. For example, Rule[$\sqsubseteq_T$] turns to $\texttt{Rel}(a$, rdf:type, $D) \leftarrow \texttt{Rel}(C$, rdfs:subClassOf, $D), \texttt{Rel}(a$, rdf:type, $C)$.

As such, given a DL KB $\Sigma =< \mathcal{T}, \mathcal{A} >$, the DL2DB system Γ w.r.t. Σ appears as a Datalog program P. The IDB of P consists of those inference rules (except Rule[$\in$]) in Γ rewritten in triples. The EDB of P is exactly an RDF DB w.r.t. Σ storing triples for the original ABox $\mathcal{A}$ and the inferred TBox $\mathcal{T}^*$, driven by Rule[$\in$]. With the help of DL reasoners, we regard the computation of TBoxes as a preprocessing. It is $\mathcal{T}^*$ instead of $\mathcal{T}$ that plays a role in this program. Since Datalog has P-complete data complexity [7], query answering in Γ is polynomial in size of the KB $\Sigma^* =< \mathcal{T}^*, \mathcal{A} >$.

Grounding a Datalog program P on the defined RDF DB needs $k{\cdot}N^M$ binding operations in maximum, where k, M and N are the number of Datalog rules, variables and constants, resp. In the case of our DL2DB system Γ, we have $k = 7$ Datalog rules excluding Rule[$\in$], each of which has maximally $M = 5$ variables acting as sub, pre, obj. The number N of constants counts those IDs representing individuals, DL classes and DL properties in Σ. Consequently, interpreting Γ in Datalog rules has a computational cost of $O(7 \cdot n^5)$.

5.3 Strategies

Although, a theoretical complexity is tractable, the evaluation strategy in practice is another story. For example, a cyclic DL TBox of $\exists P.C \sqsubseteq C$ indicates the query of $C(x)$ is based on that of $\exists P.C(x)$ which relies on $C(y)$ and $P(x, y)$. The backward chaining becomes $C(x) \leftarrow P(x, y), C(y)$, getting entangled in the recursive retrieval of C. Meanwhile, a pitfall exists when using forward chaining freely in this example. Thinking about an ABox of $P(a, a)$ and $C(a)$, we receive an infinite series of $\exists P.C(a)$, $\exists P.\exists P.C(a)$, $\exists P.\exists P.\exists P.C(a)$, and so on. A straightforward top-down implementation can take exponential time using these rules while a bottom-up approach is faster (generally polynomial), but still wastes time exploring other rules which are never used in the solution of the query [3].

To improve performance, we can use a procedure which collects subgoals of a given query top-down firstly, and then evaluates all subgoals bottom-up. Referring to techniques involved in Deductive Databases [21], the repetition of computing is by all means avoided on the one hand, and the irrelevant goals may as well be ignored on the other hand.

We are now ready to define algorithms, as shown in Table 1. To improve legibility, Rule[*] is presented, where * is the placeholder of every inference rule introduced in the DL2DB system, such as Rule[$\exists$] and Rule[$\sqsubseteq_T$], except for Rule[$\in$]. For computing the answers to a subgoal g over the KB $\Sigma^* =< \mathcal{T}^*, \mathcal{A} >$, we first exploit the relational DB to obtain $\mathcal{A}(g)$ which means a ground base of g asserted in $\mathcal{A}$, and then answers are propagated until reaching a fixpoint.

In fact, this proposed strategy, to simulate top-down semantics in a bottom-up framework, is not new, tracing back to the Magic Set [3] developed in Deductive Databases. Not providing a general magic set transformation, we regard the collection of subgoals as a magic set s.t. Ruel[$\exists$], for example, appears as $\texttt{Rel}(a,$ rdf:type, $\exists P.C) \leftarrow \texttt{Magic}(\exists P.C), \texttt{Rel}(a, P, b), \texttt{Rel}(b,$ rdf:type, $C)$. Unless required, such class expression $\exists P...\exists P.C$ as mentioned above would not be generated by the Rule[$\exists$] with `Magic`.

Table 1. Algorithms of TopDown and BottomUp

Algorithm	TopDown($q, \mathcal{T}^*$)	ButtomUp($S,\ \mathcal{T}^*, \mathcal{A}$)
Input	A query q and an inferred TBox $\mathcal{T}^*$	A set S of subgoals for q and an inferred TBox $\mathcal{T}^*$ and an original ABox $\mathcal{A}$
Output	A set S of subgoals for q	A set Ans of answers to subgoals in S

Steps	$S := \{q\}$; $top := 0$;
	while ($top <$ `sizeOf`(S)) **do**
	for each g in S **do**
	if g matches the consequent of an instantiated Rule[*]
	then $top := top + 1$; $S := S \cup$
	{those antecedents of Rule[*]};
	return S
		$Ans := \{\mathcal{A}(g) \vert g \in S\}$
		do $Ans' := Ans$
		if an instantiated Rule[*] is settled with its antecedents in Ans'
		then $Ans' := Ans' \cup$
		{the consequent of Rule[*]}
		while $Ans = Ans'$;
		return Ans

Meanwhile, there are various algorithms to address the termination of top-down methods, most of which may be considered as variants of OLDT-resolution [21]. Being a representative and used in XSB Prolog [1], the Tabling method declares tabled predicates (manually or automatically) whose evaluation is by means of a so-called SLG resolution, while non-tabled predicates are resolved as normal, i.e., using the SLD resolution steps, by which the termination reaches without infinite loops. However, complex and large DL TBoxes expect tabled predicates declared automatically. Only to exploring all instantiated rules by a 'compilation' of the TBox, those predicates are detected. It means, a top-down-like static analysis in Tabling looks similar to the collection of subgoals in our approach. Besides, we observe that, in Tabling, the table entry associated with calls to tabled predicates is enriched by inserting new derived answers, step by step. Such a gradual insertion is executed also similarly in our bottom-up computation for obtained subgoals. Briefly, running in our strategy seems not more expensive than in Tabling, equipped with physical optimizations.

So far, we believe that a bi-directional strategy is suitable for query answering in a DL-driven Datalog program. Specifically, the (indirect) cyclic DL TBox is legal which leads to the recursion unavoidable, while the superfluous computation (particularly encountering voluminous data in the Semantic Web) has been cut down by the collection of subgoals. Finally, our bi-directional strategy will terminate for a given DL KB, provided by the facts that

1. A finite number of subgoals is encountered, whose worst case is the collection of all DL classes and DL properties appearing in the KB, and

2. Each of these subgoals has a finite number of instances with the maximum of all individuals in the KB.

5.4 Comparisons

In this section, we summarize and compare systems and approaches for rule-based ontology query answering.

Knowledge compilation, making implicit information explicit in advance, has been widely used in some systems. For example, OWLIM [15], being a semantic repository, builds a materialized RDF database with the inferred closure of an OWL DLP KB. Similarly, Minerva[1] performs entailment rules bottom-up for the DL ABox inferences, plugged with a DL reasoner for precomputing the DL TBox subsumptions [16][29]. Creating DB views has been preprocessed in DLDB-OWL [20], of which each view stands for a DL class. Briefly, this kind of tools fills in DBs with data by a bottom-up precompilation, pushing a direct retrieval to back-end databases, but at the risk of a whole re-computation when updating.

Querying on-the-fly during reasoning alleviates the suffering caused by updating. There are two frameworks, KAON2 [19] and DL-lite [6], getting deserved attention. KAON2 reduces a DL KB to a disjunctive Datalog program, using the magic set algorithm developed for disjunctive programs. DL-lite is less expressive, although, any query is expressible as the union of conjunctive queries in SQL via a KB normalization, making SQL engines responsible to return the exact answers. Thus, our DL2DB stands in the middle of KAON2 and DL-lite. Generally speaking, this kind of tools follows a solid logical language, to address the challenges of expressivity and reasoning power.

It leaves a straightforward way, in which existing (top-down, bottom-up) rule engines are borrowed as a whole. In this trend, a production rule engine Jess is used by OWLJessKB[2] and OWL2Jess [17], while XSB Prolog is preferred in FLORA-2[3] and TRIPLE[4]. Also, SWI-Prolog[5] provides a Semantic Web Library dealing with the RDF data extracted from RDF(S) and OWL documents. Taking negation into account, deduction in ontologies via ASP (Answer Set Programming) has been discussed in [23] but regarding classical negation $\neg$ as default `not`, and `HEX`-Programs [11] provide a hybrid platform integrating ASP rules with external DL atoms, both of which utilize underlying ASP rule engines.

6 Discussions

Not surprising, user-defined rules are expected in real applications. Viewing those inference rules in a system level, we envision a common platform with support for SPARQL query answering in SWRL. Learning from our preliminary experiments, performance problems (e.g. running time and memory space) are discussed below, towards the development of a powerful Semantic Web tool.

6.1 Extensions

SWRL [28] is a combination of OWL DL and unary/binary Datalog rules. Technically, let S be a SWRL knowledge base, where O_C is a set of OWL classes,

[1] http://www.alphaworks.ibm.com/tech/semanticstk
[2] http://edge.cs.drexel.edu/assemblies/software/owljesskb/
[3] http://flora.sourceforge.net/
[4] http://triple.semanticweb.org/
[5] http://www.swi-prolog.org/

O_P is a set of OWL properties, and S_T is a set of OWL individuals and SWRL variables. A SWRL rule has the form: $h_1 \wedge \cdots \wedge h_n \leftarrow b_1 \wedge \cdots \wedge b_m$, where $h_i, b_j, 1 \leq i \leq n, 1 \leq j \leq m$ are atoms of the form $C(t)$ with $C \in O_C$ and $t \in S_T$, or atoms of the form $P(t_1, t_2)$ with $P \in O_P$ and $t_1, t_2 \in S_T$.

We are more interested in SWRL rules with a unique head atom, and rewriting techniques are still suitable s.t. each SWRL rule appears as: $\texttt{Rel}(s_0, p_0, o_0) \leftarrow \texttt{Rel}(s_1, p_1, o_1) \cdots \texttt{Rel}(s_n, p_n, o_n)$. Facing to a rule body having $\texttt{Rel}(a$, rdf:type, $\exists P.C)$, for instance, a rewritten version of Rule[$\exists$] as mentioned above is applied to obtain corresponding entailments, also Rule[$\sqsubseteq_T$] and others are applicable. Thus, assuming that a Datalog program works for our DL2DB system, the situation is not aggravated when more rules, in the same style, are involved. Being a preliminary work, our prototype is towards support for query answering with SWRL rules, whose semantics appeals, however, for integrity constraints.

A large variety of features have been captured in MKNF-DL [10], such as default rules and epistemic queries. Our future work includes a more general formalization concerning non-monotonic logics and latest work in [8] [11] [18] is well deserved studying.

6.2 Preliminary Experiments

We conducted initial experiments on a DBMS-based OWL repository, Minerva (DB2 in experiments). Test data sets are from the extended LUBM [16], an OWL ontology including 69 atomic classes, 39 intersection classes, 10 existential restrictions, 55 properties, 263 class subsumptions, 16 property subsumptions, 46450 class assertions, 239933 property assertions, and 25461 individuals. These documents involve one terminology file of size 66 KB, one university file of size 207 KB and twenty department files each of which is about 1300 KB.

Not tangling with complex DL class descriptions, such queries as finding instances typed of "Organization" receive answers (counting to 229 none of which is asserted in the ABox) in 0.91 second. However, the query to "Chair", defined as Person $\sqcap \exists$ headOf.Department where Department$\sqsubseteq$Organization, requires the evaluation of "Person" firstly. There are 35 classes being recognized as the subclasses of "Person", inducing totally 90 subgoals. Finally, it counts to 14995 persons (none exists in the ABox) and 40 chairs (half asserted, half inferred) in 29.73 second. As for a certain individual, who is asserted as "Man" and "FullProfessor" in the ABox, our engine further knows him typed of "Professor", "Faculty", "Employee", "Person" and "Chair" in 21.48 second.

We found that how to process intermediate results obtained from subgoals becomes a bottleneck of performance, and the worse case reports "out of memory" if temporary results are carried out in memory. Inserting them into DBs is considered, but it takes minutes since various indexes need to be established and DB needs to write logs. Thus, we turn to declared global temporary tables (without logging in DB2), in which inferred results are cached. At running time, top-down and bottom-up procedures proceed, followed by inserting intermediate answers into temporary tables active in a session. SQL engines serve for the final retrieval, concerning unions of conjunction queries on arbitral RDF triples,

from temporary tables (filled by inferred results) together with other physical DB tables.

7 Conclusions

On the Semantic Web, SPARQL is for query answering in the RDF community. An OWL ontology is RDF-based, adopting DL as its logical foundation. Given a DL KB Σ, by introducing integrity constraints inspired by MKNF-DL [10], we present its logically equivalent version, namely the DL2DB KB Σ', in the sense of query answering. Meanwhile, an inference system, the DL2DB system Γ w.r.t. Σ, takes effect, while preserving sound and complete with Σ' for non-epistemic queries. The appearance of a Datalog program moves Γ into practice, getting SPARQL queries solved.

Our proposal, to some extent, is not beyond DLP, where DLP has syntactical expressive restrictions while DL2DB has semantical integrity constraints. We still believe this paper, in an epistemic perspective, generalizes those using rules to perform OWL reasoning. As a preliminary implementation, an engine, coupled with a scalable OWL storage (e.g. Minerva [29] but not committing its ABox inferences), is developed. Answers to SPARQL queries are received in seconds on a data set of a medium size. A better performance is expected by using more optimization techniques from existing Datalog engines (e.g. [21]) and other Semantic Web applications (e.g. [22]).

References

1. *The XSB System Version 2.7.1 Volume 1: Programmer's Manual.*
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
3. François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proceedings of the 5th ACM Symposium on Principles of Database Systems*, pages 1–15, 1986.
4. Sean Bechhofer, Ian Horrocks, and Daniele Turi. The OWL Instance Store: System Description. In *Proceedings of CADE-20*, LNCS 3632, pages 177–181, 2005.
5. Francois Bry, Norbert Eisinger, Heribert Schutz, and Sunna Torge. SIC: Satisfiability Checking for Integrity Constraints. In *Proceeding of Deductive Databases and Logic Programming*, pages 25–36, 1998.
6. 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*, pages 602–607, 2005.
7. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
8. Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. On Representational Issues About Combinations of Classical Theories with Nonmonotonic Rules. In *Proceedings of KSEM*, LNCS 4092, pages 1–22, 2006.

9. Jos de Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning on the Semantic Web. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, 2005. ACM.
10. Francesco M. Donini, Daniele Nardi, and Riccardo Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.
11. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In *Proceedings of ESWC*, LNCS 4011, pages 273–287, 2006.
12. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the 12th International World Wide Web Conference*, pages 48–57. ACM, 2003.
13. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restrictions. LTCS-Report 99-08, RWTH Aachen, Germany, 1999.
14. Ian Horrocks and Sergio Tessaris. Querying the Semantic Web: a Formal Approach. In *Proceedings of ISWC*, LNCS 2342, pages 177–191, 2002.
15. Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. OWLIM: A Pragmatic Semantic Repository for OWL. In *Proceeding of International Workshop on WISE*, LNCS 3807, pages 182–192, 2005.
16. Li Ma, Yang Yang, Zhaomin Qiu, Guotong Xie, Yue Pan, and ShengPing Liu. Towards A Complete OWL Ontology Benchmark. In *Proceedings of ESWC*, LNCS 4011, pages 124–139, 2006.
17. Jing Mei, Elena Paslaru Bontas, and Zuoquan Lin. OWL2Jess: A Transformational Implementation of the OWL Semantics. In *Proceedings of International Workshops on ISPA*, LNCS 3759, pages 599–608, 2005.
18. Boris Motik and Riccardo Rosati. Closing Semantic Web Ontologies. Technical report, University of Karlsruhe, May 2006. http://kaon2.semanticweb.org/.
19. Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics*, 3(1):41–60, 2005.
20. Zhengxiang Pan and Jeff Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Practical and Scalable Semantic Systems*, 2003.
21. Kotagiri Ramamohanarao and James Harland. An introduction to deductive database languages and systems. *The VLDB Journal*, 3(2):107–122, 1994.
22. Edna Ruckhaus, Eduardo Ruiz, and Maria-Esther Vidal. Query Evaluation and Optimization in the Semantic Web. In *Proc. of ALPSWS*, 2006.
23. Terrance Swift. Deduction in Ontologies via ASP. In *Proceedings of Logic Programming and Nonmonotonic Reasoning*, LNCS 2923, 2004.
24. W3C. OWL: Web Ontology Language. http://www.w3.org/TR/owl-absyn/.
25. W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/.
26. W3C. Rule Interchange Format WG. http://www.w3.org/2005/rules/wg.
27. W3C. SPARQL Query Language. http://www.w3.org/TR/rdf-sparql-query/.
28. W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/.
29. Jian Zhou, Li Ma, Qiaoling Liu, Lei Zhang, Yong Yu, and Yue Pan. Minerva: A Scalable OWL Ontology Storage and Inference System. In *Proceedings of Asia Semantic Web Conference*, To appear, 2006.