Bridging the Gap Between OWL and Relational Databases*

Boris Motik
University of Manchester
Manchester, UK

Ian Horrocks
University of Manchester
Manchester, UK

Ulrike Sattler
University of Manchester
Manchester, UK

ABSTRACT

Schema statements in OWL are interpreted in a different way from similar statements in a relational database setting. This can lead to problems in data-centric applications, where OWL's interpretation of the statements intended as constraints may be confusing and/or inappropriate. We propose an extension of OWL that attempts to mimic the intuition behind integrity constraints in relational databases. We discuss the algorithms for checking constraint satisfaction for different types of knowledge bases, and show that, provided the constraints are satisfied, we can disregard them while answering a broad range of positive queries.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: OWL

Keywords

Semantic Web, Relational Databases, OWL

1. INTRODUCTION

The Web Ontology Language (OWL) is the W3C standard language for modeling ontologies in the Semantic Web. The logical underpinning for OWL is provided by description logics (DLs)—a family of knowledge representation formalisms with well-understood semantics and computational properties. OWL can be seen as an expressive schema language that can be used to provide flexible access to data. Schema statements in OWL are, however, interpreted in a different way to similar statements in a relational database setting. This can lead to problems in data-centric applications, where OWL's interpretation of the statements intended as constraints may be confusing and/or inappropriate.

The following example illustrates the nature of the problem. An application for managing tax returns may require each person to have a social security number. In a relational database, this would be captured by an inclusion dependency stating that, for each person, a social security number must exist. During database updates, such a dependency is interpreted as a check: whenever a person is added to the database, a check is performed to see whether that person's social security number has been specified as well; if not, the update is rejected. A similar dependency can be expressed in OWL by means of an existential restriction, but will result in a quite different behavior: adding a person without a social security number to the ontology does not raise an error, but only leads to the inference that the person in question has some unknown social security number.

In Section 2, we study such problems in detail by comparing OWL with relational databases w.r.t. their schema languages, schema and data reasoning problems, and constraint checking. Our analysis suggests that OWL ontologies can be understood as incomplete databases [15]. Many databases encountered in practice are, however, complete. To obtain a flexible schema language, one would ideally like to be able to explicitly control the degree of incompleteness. Our analysis suggests that integrity constraints (ICs)—formulae that check whether all necessary information has been explicitly provided—can be used for this purpose.

Extending logic-based KR formalisms with database-like ICs has already been considered in the literature. In his seminal paper, Reiter observed that ICs are not objective sentences about the world, but are sentences about the state of the database, and are therefore of an epistemic nature [23]. Based on this idea, constraints can be added to OWL using nonmonotonic modal [11] or rule extensions [24, 20]. Such ICs, however, can be understood as queries constraining the shape of an ABox; thus, they are not a part of the core conceptual model and do not affect schema reasoning at all. In contrast, ICs have a dual role in relational databases: they describe the possible worlds as well as the states of the databases [1]. Hence, they are used both during data reasoning (e.g., in checking the integrity of a database) and schema reasoning (e.g., in computing query subsumption). The semantic relationship between these two roles of ICs is clear, which significantly simplifies their usage.

In Section 3, we propose an extension of OWL to *extended DL knowledge bases*. Such knowledge bases allow the modeler to designate a subset of TBox axioms as ICs. For schema (TBox) reasoning, these axioms are treated as usual. For data (ABox) reasoning, however, these axioms do not derive new information; instead, they are interpreted as checks. Thus, in contrast to existing proposals, our approach provides a clear semantic relationship between the roles that ICs play during TBox and ABox reasoning. In fact, we show that, provided that an ABox satisfies the ICs, we can disregard the ICs while answering positive ABox queries. In Section 5, we discuss the typical usage patterns of our approach. Finally, in Section 6, we discuss algorithms for

*This work was partially funded by the EPRSC project REOL.

WWW2007, May 8–12, 2007, Banff, Canada.
.

checking constraint satisfaction. Due to space limitations, we present only the intuition behind them; for details, please refer to [19].

We see two main benefits of our work. First, our notion of ICs provides a solution to a common problem of applications that use OWL. Through constraint satisfaction checking, these applications can ensure that all required data has explicitly been specified in an ontology, and thus detect potential errors in the data. Second, we expect significant performance improvements in query answering, especially in applications that would make extensive use of constraints.

We assume the reader to be familiar with basics of OWL and DLs; please refer to [3] for a thorough introduction. It is well-known that the OWL DL variant of OWL corresponds to the DL $\mathcal{SHOIN}(\mathbf{D})$. Because of that, in the rest of this paper we refer to OWL and DLs interchangeably.

2. OWL VS. RELATIONAL DATABASES

2.1 Schema Language

The schema part of a DL knowledge base is typically called a *TBox* (terminology box), and is a (possibly restricted) finite set of universally quantified implications. For example, in a TBox, one can state that each person has a social security number (SSN), that any person can have at most one SSN and, conversely, that each SSN can be assigned to at most one individual. These three statements are expressed using the following TBox axioms:

$$Person \sqsubseteq \exists hasSSN.SSN \tag{1}$$

$$Person \sqsubseteq \leq 1\, hasSSN \tag{2}$$

$$SSN \sqsubseteq \leq 1\, hasSSN^{-} \tag{3}$$

Most DLs can be seen as decidable fragments of first-order logic, so their axioms have an equivalent representation as first-order formulae, cf. [7]. For example, the axioms (1)–(3) are translated into first-order logic as follows:

$$\forall x : [Person(x) \rightarrow \exists y : hasSSN(x, y) \wedge SSN(y)] \tag{4}$$

$$\forall x, y_1, y_2 : [Person(x) \wedge hasSSN(x, y_1) \wedge hasSSN(x, y_2) \rightarrow y_1 \approx y_2] \tag{5}$$

$$\forall x, y_1, y_2 : [SSN(x) \wedge hasSSN(y_1, x) \wedge hasSSN(y_2, x) \rightarrow y_1 \approx y_2] \tag{6}$$

The schema of a relational database is defined in terms of relations and dependencies among them. Different types of dependencies have been considered, such as functional, inclusion, or join dependencies. As discussed in [1], most dependencies can be represented as first-order formulae of the form (7), where $\psi(x_1, \ldots, x_n)$ and $\xi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ are conjunctions of function-free atoms:

$$\forall x_1, \ldots, x_n : [\psi(x_1, \ldots, x_n) \rightarrow \exists y_1, \ldots, y_m : \xi(x_1, \ldots, x_n, y_1, \ldots, y_m)] \tag{7}$$

Although the expressivity of DLs underlying OWL and of relational dependencies is clearly different, the schema languages of the two are quite closely related. In fact, the formula (4) corresponds to an inclusion dependency, whereas (5) and (6) correspond to key dependencies.

2.2 Interpreting the Schema

DL TBoxes and relational schemas are interpreted according to standard first-order semantics: they distinguish the legal relational structures (i.e., the structures that satisfy all axioms) from the illegal relational structures (i.e., the structures that violate some of them). In DLs, the legal structures are called *models*, whereas in relational databases they are called *database instances*; the underlying principle is, however, the same.

There is a slight technical difference between models and database instances: models are usually allowed to be infinite, whereas database instances typically must be finite. The latter restriction is due to the fact that only finite databases can be stored in practical systems. For certain classes of dependencies, the restriction to finite structures is not really relevant: whenever an infinite relational structure satisfying the schema exists, a finite structure exists as well (this is the so-called *finite model* property). For languages such as OWL, this does not necessarily hold: some ontologies are satisfied only in infinite models [3]. Even though the complexity of finite model reasoning is, for numerous DLs, the same as the complexity of reasoning w.r.t. arbitrary models, the former is usually more involved [18] and is considered only rarely. Therefore, in the rest of this paper, we drop the restriction to finite database instances and consider models and database instances to be synonymous.

2.3 Domains and Typing

Relational databases often assign types to columns of relations; for example, the second position of *hasSSN* could be restricted to strings of a certain form. This is important in practice since the physical layout of a database is determined by the column types. In contrast, typing is not considered in many theoretical works (e.g., in algorithms for checking containment of conjunctive queries); rather, columns draw their values from a common countable domain set [1].

To provide for explicit typing of relationships, the DLs underlying OWL have *datatypes*—a simplified variant of the so-called *concrete domains* [4].

Here, we consider neither typed relational schemas nor DL knowledge bases with concrete domains, and simply interpret both relational schemata and TBoxes in first-order logic. This simplifies both formalisms significantly. For example, adding key constraints to untyped DLs is not a problem [8], whereas adding them to DLs with typed predicates is much more involved [17].

2.4 Schema Reasoning

Checking subsumption relationships between concepts has always been a central problem in DL reasoning. A concept C is subsumed by a concept D with respect to a DL TBox $\mathcal{T}$ if the extension of C is included in the extension of D in each model I that satisfies $\mathcal{T}$. This inference has many uses; for example, in the development of an OWL ontology, the entailed subsumption relationships can be used to detect modeling errors. Furthermore, concept subsumption has been used to optimize query answering [13], especially when generalized to subsumption of conjunctive queries [10, 12]. Another important TBox inference is checking concept satisfiability—that is, determining whether a model of $\mathcal{T}$ exists in which a given concept has a nonempty extension. If a concept is unsatisfiable, this is usually due to modeling errors, so this inference is also useful in ontology modeling.

Reasoning about the schema is certainly not the most prominent feature of relational databases, so it may come as a surprise that a significant amount of database research

has been devoted to it. The most important schema-related inference in databases is checking containment of conjunctive queries [1]: a query Q_1 is contained in a query Q_2 w.r.t. a schema $\mathcal{T}$ if the answer of Q_1 is contained in the answer of Q_2 for each database instance that satisfies $\mathcal{T}$. This inference is used by virtually all database systems in query optimization to rewrite queries into equivalent ones that can be answered more efficiently. Another inference often considered in the literature is dependency minimization—that is, computing a minimal equivalent schema.

In both DLs and relational databases, schema reasoning problems correspond to checking whether some formula φ holds in each model (database instance) of $\mathcal{T}$—that is, checking whether $\mathcal{T} \models \varphi$. In other words, the terminological problems in both DLs and relational databases correspond to first-order consequences of a first-order theory. Since the problems are the same, it should not come as a surprise that the methods used to solve them are closely related. Namely, reasoning in description logics is typically performed by tableau algorithms [5], whereas the state-of-the-art reasoning technique in relational databases is chase [1]. Apart from notational differences, the principles underlying these two techniques are the same: they both use a set of rules to try to construct a model that satisfies the schema $\mathcal{T}$ but not the formula φ.

To summarize, from a semantic point of view, DLs and databases treat schema reasoning problems in the same way. Therefore, DLs can be understood as rather expressive, but decidable, database schema languages. As we discuss in the following sections, the differences between DLs and databases arise when we consider database data.

2.5 Query Answering

A DL knowledge base consists of the schema part, the TBox, and a data part, the ABox. The simplest form of query answering in description logics is *instance checking*—that is, checking whether an individual a is contained in a concept C in each model of the knowledge base $\mathcal{K}$, commonly written as $\mathcal{K} \models C(a)$. Instance checking can be generalized to answering conjunctive queries over DL knowledge bases [10, 12]. Thus, a DL query can be viewed as a first-order formula φ with free variables $x_1, \ldots, x_n$. Like schema reasoning, the semantics of query answering in description logics is defined as first-order entailment; that is, it takes into account all models of $\mathcal{K}$: a tuple $a_1, \ldots, a_n$ is in the answer to φ over the knowledge base $\mathcal{K}$ if $\mathcal{K} \models \varphi[a_1/x_1, \ldots, a_n/x_n]$, where the formula $\varphi[a/x]$ is obtained by replacing in φ all free occurrences of x with a.

Queries in relational databases are first-order formulae (restricted in a way to make them domain independent) [1], so they are similar to queries in description logics. A significant difference between DLs and relational databases is the way in which the queries are evaluated. Let φ be a first-order formula with the free variables $x_1, \ldots, x_n$. A tuple $a_1, \ldots, a_n$ is an answer to φ over a database instance I if $I \models \varphi[a_1/x_1, \ldots, a_n/x_n]$. Hence, unlike in description logics, query answering in relational databases does not consider all databases instances that satisfy the knowledge base $\mathcal{K}$; instead, it considers only the given instance I. In other words, query answering in relational databases is not defined as entailment, but as model checking, where the model is the given database instance.

Although the definition of query answering in relational databases from the previous paragraph is the most widely used one, a significant amount of research has also been devoted to answering queries over incomplete databases [15]—a problem that is particularly interesting in information integration. An incomplete database $\mathcal{K}$ is described by a set $\mathcal{A}$ of incomplete extensions of the schema relations and a set $\mathcal{T}$ of dependencies specifying how the incomplete extensions relate to the actual database instance. Queries in incomplete databases are also (possibly restricted) first-order formulae. In contrast to complete databases, a tuple $a_1, \ldots, a_n$ is a *certain* answer to φ over $\mathcal{K}$ if $I \models \varphi[a_1/x_1, \ldots, a_n/x_n]$ for each database instance I that satisfies $\mathcal{A}$ and $\mathcal{T}$. In other words, query answering in incomplete databases is defined as first-order entailment, exactly as in description logics. Consequently, DL query answering can be understood as answering queries in incomplete databases.

2.6 Constraint Checking

In DLs, one can check whether an ABox $\mathcal{A}$ is consistent with a TBox $\mathcal{T}$—that is, whether a model I of both $\mathcal{A}$ and $\mathcal{T}$ exists—and thus detect possible contradictions in $\mathcal{A}$ and $\mathcal{T}$. This inference is not, however, a suitable basis for constraint checking. For example, let $\mathcal{T}$ contain the axioms (1)–(3), and let $\mathcal{A}$ contain only the following axiom:

$$Person(Peter) \tag{8}$$

If (1) were interpreted as an integrity constraint, we would expect it to be violated by $\mathcal{A} \cup \mathcal{T}$: the ABox states that *Peter* is a person without specifying his social security number. The knowledge base $\mathcal{A} \cup \mathcal{T}$ is, however, satisfiable: the axiom (1) is not interpreted as a check; rather, it implies the existence of some (unknown) SSN. Thus, $\mathcal{T}$ describes the legal models, but not the legal ABoxes. In fact, the DLs underlying OWL do not provide any means to express database-like integrity constraints.

In contrast, constraints play a central role in relational databases, where they are used to ensure the integrity of data. As in DLs, a relational schema $\mathcal{T}$ is a set of formulae that must hold for any database instance. A relational database instance, however, corresponds to exactly one model, whereas a DL ABox typically has infinitely many models. Just like query answering, constraint checking in relational databases corresponds to model checking: given a database instance I, a relational database checks the satisfaction of the schema constraints $\mathcal{T}$ by checking whether $I \models \mathcal{T}$.[1] In our example, if the ABox $\mathcal{A}$ is taken as the database instance I, then, clearly, $I \not\models \mathcal{T}$—as expected, the integrity constraints are not satisfied.

2.7 Discussion

OWL users have complained about the "open-world semantics of OWL" or "the unintuitive constraints in OWL." From our experience, this is often due to incorrect interpretation of the fundamental assumptions behind OWL and relational databases, respectively.

From the standpoint of conceptual modeling, description logics provide a very expressive, but still decidable language that has proven to be implementable in practice. The open-world semantics is natural for a schema language since a schema defines the structure of all legal database instances.

[1]In practice, constraints are incrementally checked after database updates; these dynamic aspects are, however, not important for this discussion.

When computing the subsumption relationship between concepts or queries, we do not have a fixed instance at all. Therefore, we cannot employ the closed-world assumption in the traditional sense, but have to interpret $\mathcal{T}$ under open-world semantics.

Complaints from users are most common when they relate to the use of OWL in data-centric applications—that is, applications that focus on the management of large volumes of data. In practice, relational databases are typically complete: any missing information is either encoded metalogically (e.g., users often include fields such as *hasSpecifiedSSN* to signal that particular data has been supplied in the database), or it is represented by *null-values* (which are also interpreted outside first-order logic). In contrast, ABoxes in DLs are closely related to incomplete (relational) databases. Clearly, problems may arise if certain aspects of the information about individuals in ABoxes are expected to be complete. To understand the problems that occur in such cases, consider the following example. Biopax[2] is an ontology developed by the bioinformatics community with the goal of facilitating data integration and interchange between biological databases. It defines a property *NAME* and states its domain to be the union of *bioSource*, *entity*, and *dataSource*:

$$\exists NAME.\top \sqsubseteq bioSource \sqcup entity \sqcup dataSource \tag{9}$$

The intention behind this axiom is to define which objects can be named—that is, to ensure that a name is stated only for objects of the appropriate type. In fact, the data in the Biopax ontology is complete w.r.t. this constraint: each object with a name is also typed (sometimes indirectly through the class hierarchy) to at least one of the required classes. The axiom (9), however, does not act as a constraint; instead, it says that, if some object has a name, then it can be *inferred* to be either a *bioSource*, an *entity*, or a *dataSource*. Therefore, (9) cannot be used to check whether all data is correctly typed. Furthermore, since the concept in the consequent is a disjunction, the axiom (9) requires reasoning by case, which is one of the reasons for intractability of DL reasoning algorithms [3, Chapter 3]. Hence, the axiom (9) causes two types of problems: on the one hand, it does not have the desired semantics and, on the other hand, it introduces a performance penalty for reasoning.

Representing incomplete information is, however, needed in many applications. Consider the following axiom stating that married people are eligible for a tax cut:

$$\exists marriedTo.\top \sqsubseteq TaxCut \tag{10}$$

To apply this axiom, we do not necessarily need to know the name of the spouse; we only need to know that a spouse exists. Thus, we may state the following fact:

$$(\exists marriedTo.Woman)(Peter) \tag{11}$$

We are now able to derive that *Peter* is eligible for a tax cut even without knowing the name of his spouse. Providing complete information can be understood as filling in a "Spouse name" box on a tax return, whereas providing incomplete information can be understood as just ticking the "Married" box. The existential quantifier can be understood as a well-behaved version of null-values that explicitly specifies the semantics of data incompleteness. For use cases that require reasoning with incomplete information, description logics provide a sound and a well-understood foundation.

Thus, one would ideally like to be able to explicitly control "the amount of incompleteness" in an ontology. Such a mechanism would allow the ontology modeler to explicitly state which data must be fully specified and which can be left incomplete. This goal can be achieved through an appropriate form of integrity constraints that can check whether all data has been specified as required by the ontology. Transforming inappropriate and/or erroneously introduced axioms into integrity constraints should also speed up query answering by eliminating unintended and potentially complex inferences.

3. CONSTRAINTS FOR OWL

In this section, we extend DL knowledge bases with integrity constraints in order to overcome the problems discussed in the previous section. Since TBoxes are first-order formulae, it would be straightforward to apply the model checking approach described in Section 2 to OWL. In such an approach, an ABox would be interpreted as a single model and the TBox axioms as formulae that must be satisfied in a model; the constraints would be satisfied if $\mathcal{A} \models \mathcal{T}$. Such an approach is not satisfactory, however, as it would be an "all-or-nothing" choice: it would assume that *all* information in the ABox is complete; furthermore, TBox axioms would only be used to check whether an ABox is of an appropriate form and could not imply new facts. To obtain a more useful formalism, we propose a combination of inferencing and constraint checking. For example, let $\mathcal{A}_1$ be the following ABox:

$$Student(Peter) \tag{12}$$
$$hasSSN(Peter, nr12345) \tag{13}$$
$$SSN(nr12345) \tag{14}$$
$$Student(Paul) \tag{15}$$

Furthermore, let $\mathcal{T}_1$ be the following TBox:

$$Student \sqsubseteq Person \tag{16}$$
$$Person \sqsubseteq \exists hasSSN.SSN \tag{17}$$

Let us assume that we want to treat (17) as a constraint, but (16) as a normal axiom. Then, we derive $Person(Peter)$ and $Person(Paul)$ by (16). The axiom (17) is satisfied for *Peter* due to (12), (13), and (14); however, an SSN has not been specified for *Paul*, so we would expect (17) to be violated.

Following this intuition, we define extended DL knowledge bases to distinguish the axioms that imply new facts from the axioms that check whether the necessary information is present. The following definition is applicable to any DL, so it can be used with OWL as well.

DEFINITION 3.1. *An* extended DL knowledge base *is a triple* $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ *such that*

- $\mathcal{S}$ *is a finite set of* standard *TBox axioms,*
- $\mathcal{C}$ *is a finite set of* constraint *TBox axioms, and*
- $\mathcal{A}$ *is a finite set of ABox assertions* $(\neg)A(a)$, $R(a,b)$, $a \approx b$, *or* $a \not\approx b$, *for* A *an atomic concept,* R *a role, and* a *and* b *individuals.*

[2]http://www.biopax.org/

ABoxes in which only (possibly negated) atomic concepts are allowed are said to be *extensionally reduced*. Assuming that the ABox is extensionally reduced does not result in any loss of generality because $\mathcal{S}$ can be used to introduce names for arbitrary concept expressions.

Next, we consider how to define an appropriate semantics for extended DL knowledge bases. The simplest solution is to interpret $\mathcal{A} \cup \mathcal{S}$ in the standard first-order way and require $\mathcal{C}$ to be satisfied in each model I such that $I \models \mathcal{A} \cup \mathcal{S}$. The following example, however, shows that this does not satisfy our intuition. Let $\mathcal{A}_2$ contain only the fact (12), $\mathcal{S}_2 = \emptyset$, and let $\mathcal{C}_2$ contain only the axiom (17). The interpretation $I = \{Student(Peter), Person(Peter)\}$ is a model of $\mathcal{A}_2 \cup \mathcal{S}_2$ that does not satisfy $\mathcal{C}_2$, which would make $\mathcal{C}_2$ not satisfied for $\mathcal{A}_2 \cup \mathcal{S}_2$. Intuitively, though, the fact $Person(Peter)$ is not implied by $\mathcal{A}_2 \cup \mathcal{S}_2$, so we should not check whether *Peter* has an SSN at all; $\mathcal{C}_2$ should hold only for the facts that are implied by $\mathcal{A}_2 \cup \mathcal{S}_2$.

These considerations might suggest that $\mathcal{C}$ should hold for all first-order consequences of $\mathcal{A} \cup \mathcal{S}$. In the example from the previous paragraph, this produces the desired behavior: $Person(Peter)$ is not a consequence of $\mathcal{A}_2 \cup \mathcal{S}_2$, so the axiom from $\mathcal{C}_2$ should not be checked for *Peter*. Consider, however, the ABox $\mathcal{A}_3$ containing only the following axiom:

$$Cat(ShereKahn) \tag{18}$$

Furthermore, let $\mathcal{S}_3$ be the standard TBox containing the following axiom:

$$Cat \sqsubseteq Tiger \sqcup Leopard \tag{19}$$

Finally, let $\mathcal{C}_3$ be the constraint TBox containing the following two axioms:

$$Tiger \sqsubseteq Carnivore \tag{20}$$
$$Leopard \sqsubseteq Carnivore \tag{21}$$

Now neither $Tiger(ShereKahn)$ nor $Leopard(ShereKahn)$ is a first-order consequence of $\mathcal{A}_3 \cup \mathcal{S}_3$, which means that the axioms from $\mathcal{C}_3$ are satisfied; furthermore, we have

$$\mathcal{A}_3 \cup \mathcal{S}_3 \not\models Carnivore(ShereKahn).$$

This answer does not satisfy our intuition: in each model of $\mathcal{A}_3 \cup \mathcal{S}_3$, either $Tiger(ShereKahn)$ or $Leopard(ShereKahn)$ holds, but $Carnivore(ShereKahn)$ does not necessarily hold in either case; hence, by treating (20)–(21) as constraints and not as standard axioms, we neither get a constraint violation nor derive the consequence $Carnivore(ShereKahn)$.

Intuitively, the constraints should check whether the facts derivable from $\mathcal{A} \cup \mathcal{S} \cup \mathcal{C}$ are derivable from $\mathcal{A} \cup \mathcal{S}$ only. This notion seems to be nicely captured by minimal models; that is, we check $\mathcal{C}$ only in the *minimal models* of $\mathcal{A} \cup \mathcal{S}$. Roughly speaking, a model I with an interpretation domain $\triangle^I$ of a formula φ is minimal if each interpretation I' over $\triangle^I$ such that $I' \subset I$ is not a model of φ, where we consider an interpretation to be represented as the set of all positive ground facts that are true in it. Consider again $\mathcal{A}_2$, $\mathcal{S}_2$, and $\mathcal{C}_2$. The fact $Person(Peter)$ is not derivable from $\mathcal{A}_2 \cup \mathcal{S}_2$ in any minimal model (in fact, there is only a single minimal model), so the constraint axiom (17) is not violated. In contrast, $\mathcal{A}_3 \cup \mathcal{S}_3$ has exactly two minimal models:

$$\begin{aligned} I_1 &= \{Cat(ShereKahn), Tiger(ShereKahn)\} \\ I_2 &= \{Cat(ShereKahn), Leopard(ShereKahn)\} \end{aligned}$$

These two models can be interpreted as the minimal sets of derivable consequences. The constraint TBox $\mathcal{C}_3$ is not satisfied in all minimal models (in fact, it is violated in each of them). In contrast, let $\mathcal{A}_4 = \mathcal{A}_3$ and $\mathcal{C}_4 = \mathcal{C}_3$, and let $\mathcal{S}_4$ contain the following axiom:

$$Cat \sqsubseteq (Tiger \sqcap Carnivore) \sqcup (Leopard \sqcap Carnivore) \tag{22}$$

Now $Carnivore(ShereKahn)$ is derivable whenever we can derive either $Tiger(ShereKahn)$ or $Leopard(ShereKahn)$, so the constraints should be satisfied. Indeed, $\mathcal{A}_4 \cup \mathcal{S}_4$ has the following two minimal models:

$$\begin{aligned} I_3 &= I_1 \cup \{Carnivore(ShereKahn)\} \\ I_4 &= I_2 \cup \{Carnivore(ShereKahn)\} \end{aligned}$$

Both I_3 and I_4 satisfy $\mathcal{C}_4$. Furthermore, we do not lose any consequences, despite the fact that we treat (20)–(21) as constraints, since the following holds:

$$\mathcal{A}_4 \cup \mathcal{S}_4 \models Carnivore(ShereKahn)$$

The mentioned definition of minimal models has been used, with minor differences, in an extension of DLs with circumscription [6] and in the semantics of open answer set programs [14]. Consider, however, the following ABox $\mathcal{A}_5$:

$$Woman(Alice) \tag{23}$$
$$Man(Bob) \tag{24}$$

Furthermore, let $\mathcal{S} = \emptyset$ and $\mathcal{C}$ contain the following axiom:

$$Woman \sqcap Man \sqsubseteq \bot \tag{25}$$

No axiom implies that *Alice* and *Bob* should be interpreted as the same individual, so we expect them to be different "by default" and the constraint to be satisfied. Now our problem is that we must consider all interpretation domains, so let $\triangle^I = \{\alpha\}$. Because $\triangle^I$ contains only one object, we must interpret both *Alice* and *Bob* as α. Clearly, $I = \{Woman(\alpha), Man(\alpha)\}$ is a minimal model of $\mathcal{A}_5$, and it does not satisfy $\mathcal{C}_5$.

This problem might be remedied by making the unique name assumption (UNA)—that is, by requiring each constant to be interpreted as a different individual. This is, however, rather restrictive and not compatible with OWL. Another solution is to interpret $\mathcal{A} \cup \mathcal{S}$ in a Herbrand model (that is, a model in which each constant is interpreted by itself) where $\approx$ is a congruence relation; then, we minimize the interpretation of $\approx$ together with all the other predicates. In such a case, the only minimal model of $\mathcal{A}_5$ is $I' = \{Woman(Alice), Man(Bob)\}$ (the extension of $\approx$ is minimized, and it is empty), and $\mathcal{C}_5$ is satisfied in I'.

Unfortunately, existential quantifiers pose a whole range of problems for constraints. Let $\mathcal{A}_6$ contain these axioms:

$$HasChild(Peter) \tag{26}$$
$$HasHappyChild(Peter) \tag{27}$$
$$TwoChildren(Peter) \tag{28}$$

Furthermore, let $\mathcal{S}_6$ contain these axioms:

$$HasChild \sqsubseteq \exists hasChild.Child \tag{29}$$
$$HasHappyChild \sqsubseteq \exists hasChild.(Child \sqcap Happy) \tag{30}$$

Finally, let $\mathcal{C}_6$ contain these constraint:

$$TwoChildren \sqsubseteq \geq 2\, hasChild.Child \tag{31}$$

It seems intuitive for $\mathcal{C}_6$ to be satisfied in $\mathcal{A} \cup \mathcal{S}_6$: no axiom in $\mathcal{S}_6$ forces the son and the daughter of *Peter*—the two individuals whose existence is implied by (29) and (30)—to be the same, so we might conclude that they are different.

Now consider the following quite similar example. Let $\mathcal{C}_7 = \mathcal{C}_6$, and let $\mathcal{A}_7$ contain the following axioms:

$$HasChild(Peter) \tag{32}$$

$$TwoChildren(Peter) \tag{33}$$

Furthermore, let $\mathcal{S}_7$ contain the following axiom:

$$HasChild \sqsubseteq \exists hasChild.Child \sqcap \exists hasChild.Child \tag{34}$$

As in the previous example, $\mathcal{C}_7$ is satisfied in $\mathcal{A}_7 \cup \mathcal{S}_7$ since (34) introduces two (possibly identical) individuals in the extension of *Child*. Let $\mathcal{S}'_7$ be a standard TBox containing only the axiom (35):

$$HasChild \sqsubseteq \exists hasChild.Child \tag{35}$$

Now $\mathcal{C}_7$ is not satisfied in $\mathcal{A} \cup \mathcal{S}'_7$ since (35) implies the existence of only one child. Given that $\mathcal{S}'_7$ is semantically equivalent to $\mathcal{S}_7$, this is rather unsatisfactory; furthermore, it suggests that $\mathcal{C}_7$ should not be satisfied in $\mathcal{A}_7 \cup \mathcal{S}_7$, since (34) requires the existence of only one individual. Recall, however, that $\mathcal{S}_6$ and $\mathcal{S}_7$ are quite closely related: the effect of (34) with respect to *Child* is the same as that of (29) and (30). Hence, if (34) should introduce only one individual, then (29) and (30) should do so as well, which is in conflict with our intuition that $\mathcal{C}_6$ should be satisfied in $\mathcal{A}_6 \cup \mathcal{S}_6$.

Thus, our intuition does not give us a clear answer as to the appropriate treatment of existential quantifiers in the standard TBox: the names of the concepts and the structure of the axioms suggest that the existential quantifiers in (29) and (30) should introduce different individuals, whereas the existential quantifiers in (34) should "reuse" the same individual. These two readings pull in opposite directions, so a choice between the two should be based on other criteria.

The previous example involving $\mathcal{S}_7$ and $\mathcal{S}'_7$ reveals an important disadvantage of one choice for the treatment of existential quantifiers: if we require each existential quantifier to introduce a distinct individual, then it is possible for a constraint TBox $\mathcal{C}$ to be satisfied in $\mathcal{A} \cup \mathcal{S}$, but not in $\mathcal{A} \cup \mathcal{S}'$, even though $\mathcal{S}$ and $\mathcal{S}'$ are semantically equivalent. As we have seen, $\mathcal{C}_7$ is satisfied in $\mathcal{A}_7 \cup \mathcal{S}_7$, but not in $\mathcal{A}_7 \cup \mathcal{S}'_7$, even though $\mathcal{S}_7$ and $\mathcal{S}'_7$ are equivalent. It is clearly undesirable for the satisfaction of constraints to depend on the syntactic structure of the standard TBox.

Concerning the alternative choice for the treatment of existential quantifiers, introducing distinct individuals for each existential quantifier seems to make checking satisfaction of constraints easier: this choice is closely related to *skolemization* [21], the well-known process of representing existential quantifiers with new function symbols. Skolemization ensures that every minimal model is forest-like (i.e., it consists of trees possibly interconnected at roots), which we use in our procedure for checking constraint satisfaction [19]. The next definition uses outer skolemization, in which the existential quantifiers are replaced with functional symbols from outermost to the innermost one.

DEFINITION 3.2. *Let φ be a first-order formula and $\mathsf{sk}(\varphi)$ the formula obtained by outer skolemization of φ [21]. A Herbrand interpretation w.r.t. φ is a Herbrand interpretation defined over the signature of $\mathsf{sk}(\varphi)$. A Herbrand interpretation I w.r.t. φ is a* model *of φ, written $I \models \varphi$, if it satisfies φ in the usual sense. A Herbrand model I of φ is* minimal *if $I' \not\models \varphi$ for each Herbrand interpretation I' such that $I' \subset I$. We write $\mathsf{sk}(\varphi) \models_{\mathsf{MM}} \psi$ if $I \models \psi$ for each minimal Herbrand model I of φ.*

We now define the notion of satisfaction of a constraint TBox for extended DL knowledge bases. Our definition relies upon an operator π that translates a set of DL axioms S into an equivalent formula $\pi(S)$ of first-order logic (possibly with equality and counting quantifiers) [3, 7].

DEFINITION 3.3. *Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base. The constraint TBox $\mathcal{C}$ is* satisfied *in $\mathcal{K}$ if $\mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S})) \models_{\mathsf{MM}} \pi(\mathcal{C})$. By an abuse of notation, we often omit π and simply write $\mathsf{sk}(\mathcal{A} \cup \mathcal{S}) \models_{\mathsf{MM}} \mathcal{C}$.*

Note that the addition of constraints does not change the semantics of DLs or OWL: Definition 3.3 is only concerned with the semantics of constraints, and a traditional knowledge base $(\mathcal{A}, \mathcal{T})$ can be seen as an extended knowledge base $(\mathcal{A}, \emptyset, \mathcal{T})$. For subsumption and concept satisfiability tests, we would use $\mathcal{S} \cup \mathcal{C}$ together as the schema, as usual. As discussed above, skolemization introduces a new function symbol for each existential quantifier, which effectively introduces a new individual for each quantifier. We invite the reader to convince himself that Definition 3.3 closely follows our intuition on the examples presented thus far. Furthermore, in Section 4 we show that, if the constraints are satisfied, we can throw them away without losing any positive consequences; that is, we can answer positive queries taking into account only $\mathcal{A}$ and $\mathcal{S}$. We take this as confirmation that our semantics of constraints is intuitive.

We now consider a nonobvious consequence of our semantics. Let $\mathcal{A}_8$ be an ABox with only the following axioms:

$$Vegetarian(Ian) \tag{36}$$

$$eats(Ian, soup) \tag{37}$$

Furthermore, let $\mathcal{S}_8 = \emptyset$, and let $\mathcal{C}_8$ contain only the following constraint:

$$Vegetarian \sqsubseteq \forall eats.\neg Meat \tag{38}$$

One might intuitively expect $\mathcal{C}_8$ not to be satisfied for $\mathcal{A}_8$, since the ABox does not state $\neg Meat(soup)$. Contrary to our intuition, $\mathcal{C}_8$ is satisfied in $\mathcal{A}_8$: the interpretation I containing only the facts (36) and (37) is the only minimal Herbrand model of $\mathcal{A}_8$ and $I \models \mathcal{C}_8$. In fact, the axiom (38) is equivalent to the following axiom:

$$Vegetarian \sqcap \exists eats.Meat \sqsubseteq \bot \tag{39}$$

When written in the latter form, the axiom should be intuitively satisfied, since $Meat(soup)$ is not derivable.

As this example illustrates, the intuitive meaning of constraints is easier to grasp if we transform them into the form $C \sqsubseteq D$, where both C and D are negation-free concepts. Such constraints allow the checking of positive facts. To check negative facts, we must give them atomic names. Let $\mathcal{A}_9 = \mathcal{A}_8$; furthermore, let $\mathcal{S}_9$ contain the following axiom:

$$NotMeat \equiv \neg Meat \tag{40}$$

Finally, let $\mathcal{C}_9$ contain the following axiom:

$$Vegetarian \sqsubseteq \forall eats.NotMeat \tag{41}$$

The constraint (40) is now of the form $C \sqsubseteq D$, so it is easier to understand the intuition behind it: everything that is eaten by an instance of *Vegetarian* should provably be *NotMeat*. Now, $\mathcal{A}_9 \cup \mathcal{S}_9$ has the following two minimal models, and $I_1 \not\models \mathcal{C}_9$, so $\mathcal{C}_9$ is not satisfied for $\mathcal{A}_9$:

$$I_1 = \{Vegetarian(Ian), eats(Ian, soup), Meat(soup)\}$$
$$I_2 = \{Vegetarian(Ian), eats(Ian, soup), NotMeat(soup)\}$$

If we add to $\mathcal{A}_9$ the fact $NotMeat(soup)$, then only I_2 is a minimal model, and $\mathcal{C}_9$ becomes satisfied as expected.

4. CONSTRAINTS AND QUERIES

We now state an important result about answering unions of positive conjunctive queries in extended DL knowledge bases: if the constraints are satisfied, we need not consider them in query answering. This indicates that our semantics for constraints is reasonable: constraints act as checks, and, if they are satisfied, we can throw them away without losing relevant consequences. Moreover, this result is practically very important because it makes query answering simpler. In Section 6, we show that, for certain types of OWL ontologies, both checking constraints and query answering can be easier than standard DL reasoning. Before proceeding, we first remind the reader of the definition of unions of conjunctive queries.

DEFINITION 4.1. *Let* $\mathbf{x}$ *be a set of* distinguished *and* $\mathbf{y}$ *a set of* nondistinguished *variables. A* conjunctive query $Q(\mathbf{x}, \mathbf{y})$ *is a finite conjunction of positive atoms of the form* $A(t_1, \ldots, t_m)$*, where* t_i *are either constants, distinguished, or nondistinguished variables.*[3] *A* union of finitely many conjunctive queries $Q_i(\mathbf{x}, \mathbf{y_i})$, $1 \leq i \leq n$, *is the formula* $\gamma(\mathbf{x}) = \bigvee_{i=1}^{n} \exists \mathbf{y_i} : Q_i(\mathbf{x}, \mathbf{y_i})$. *A tuple of constants* $\mathbf{c}$ *is* answer *to* $\gamma(\mathbf{x})$ *over a DL knowledge base* $\mathcal{K}$*, written* $\mathcal{K} \models \gamma(\mathbf{c})$*, if* $\pi(\mathcal{K}) \models \gamma(\mathbf{x})[\mathbf{c}/\mathbf{x}]$.

Our result is captured by the following theorem, whose proof can be found in [19]:

THEOREM 4.2. *Let* $\mathcal{K}$ *be an extended DL knowledge base that satisfies* $\mathcal{C}$*. Then, for any union of conjunctive queries* $\gamma(\mathbf{x})$ *over* $\mathcal{K}$ *and any tuple of constants* $\mathbf{c}$*, we have*

$$\mathcal{A} \cup \mathcal{S} \cup \mathcal{C} \models \gamma(\mathbf{c}) \text{ if and only if } \mathcal{A} \cup \mathcal{S} \models \gamma(\mathbf{c}).$$

Consider, for example, the following knowledge base. Let the standard TBox $\mathcal{S}_{10}$ contain the following axioms:

$$Cat \sqsubseteq Pet \tag{42}$$
$$\exists hasPet.Pet \sqsubseteq PetOwner \tag{43}$$

Let the constraint TBox $\mathcal{C}_{10}$ contain the following axiom:

$$CatOwner \sqsubseteq \exists hasPet.Cat \tag{44}$$

Finally, let the ABox $\mathcal{A}_{10}$ contain the following assertions:

$$CatOwner(John) \tag{45}$$
$$hasPet(John, Garfield) \tag{46}$$
$$Cat(Garfield) \tag{47}$$

Under the standard semantics, we can draw the following conclusion from $\mathcal{K}$:

$$\mathcal{S}_{10} \cup \mathcal{C}_{10} \cup \mathcal{A}_{10} \models PetOwner(John)$$

[3]The predicate A can be the equality predicate $\approx$.

Furthermore, it is easy to see that the constraint (44) is satisfied in $\mathcal{K}$: the only derivable fact about *CatOwner* is $CatOwner(John)$ and the ABox contains explicit information that *John* owns *Garfield* who is a *Cat*. Therefore, we do not really need the axiom (44) to imply the existence of the owned cat: whenever we can derive $CatOwner(x)$ for some x, we can derive the information about the cat of x as well. Hence, we can disregard (44) during query answering; even if we do that, our conclusion holds just the same:

$$\mathcal{S}_{10} \cup \mathcal{A}_{10} \models PetOwner(John)$$

Note that the entailment in Theorem 4.2 uses the standard semantics of DLs; that is, we do not assume a closed-world semantics for query answering. Furthermore, Theorem 4.2 does not guarantee preservation of negative consequences; in fact, such consequences may change, as the following example demonstrates. Let $\mathcal{S}_{11} = \emptyset$, $\mathcal{C}_{11}$ contain the axiom

$$Cat \sqcap Dog \sqsubseteq \bot \tag{48}$$

and $\mathcal{A}_{11}$ contain the axiom (47). By taking $\mathcal{S}_{11}$ into account, we get the following inference:

$$\mathcal{S}_{11} \cup \mathcal{C}_{11} \cup \mathcal{A}_{11} \models \neg Dog(Garfield)$$

Furthermore, the constraints are satisfied; however, if we disregard $\mathcal{C}_{11}$, we lose this consequence:

$$\mathcal{S}_{11} \cup \mathcal{A}_{11} \not\models \neg Dog(Garfield)$$

A similar example can be given for queries containing universal quantifiers.

Theorem 4.2 has an important implication with respect to TBox reasoning. Let $\gamma_1(\mathbf{x})$ and $\gamma_2(\mathbf{x})$ be unions of conjunctive queries such that $\pi(\mathcal{K}) \models \forall \mathbf{x} : [\gamma_1(\mathbf{x}) \rightarrow \gamma_2(\mathbf{x})]$. Provided that $\mathcal{C}$ is satisfied in $\mathcal{K}$, each answer to $\gamma_1(\mathbf{x})$ w.r.t. $\mathcal{A} \cup \mathcal{S}$ is also an answer to $\gamma_2(\mathbf{x})$ w.r.t. $\mathcal{A} \cup \mathcal{S}$. To summarize, we can check subsumption of unions of conjunctive as usual, treating $\mathcal{C} \cup \mathcal{S}$ as a usual DL TBox. Subsequently, for knowledge bases that satisfy $\mathcal{C}$, we can ignore $\mathcal{C}$ when answering queries, but query answers will still satisfy the established subsumption relationships between queries.

5. TYPICAL USAGE PATTERNS

The notion of constraints from Section 3 is very general. To provide practical guidance for modelers, we now discuss some typical usage patterns.

5.1 Participation Constraints

Participation constraints specify how objects participate in relationships. These constraints involve two concepts C and D and a relation R between them, and they state that each instance of C must participate in one or more R-relationships with instances of D. Participation constraints typically also define the cardinality of the relationship. The general form of participation constraints is as follows, where $\bowtie \in \{\leq, \geq, =\}$ and n is a nonnegative integer:

$$C \sqsubseteq \bowtie n\, R.D \tag{49}$$

Participation constraints are similar to inclusion dependencies in relational databases since they state that, for each object in one predicate, certain other objects must be present in other predicates as well.

A typical participation constraint is the axiom (1), which states that each person must have an explicitly specified

social security number. Another example is the following statement allowing each person to have at most one spouse:

$$Person \sqsubseteq \leq 1\, marriedTo.Person \tag{50}$$

To understand the difference in treating (50) as a standard axiom or a constraint, consider the following ABox $\mathcal{A}$:

$$Person(Peter) \tag{51}$$

$$marriedTo(Peter, Ann) \tag{52}$$

$$marriedTo(Peter, Mary) \tag{53}$$

If (50) were a part of the standard TBox $\mathcal{S}$, then $\mathcal{A} \cup \mathcal{S}$ would be satisfiable; furthermore, due to (50), we would derive $Ann \approx Mary$. If we put (50) into the constraint TBox $\mathcal{C}$, then the only minimal model of $\mathcal{A}$ contains exactly the facts (51)–(53). Namely, the equality predicate $\approx$ is minimized along with all the other predicates, so *Ann* is different from *Mary*. This matches our intuition because there is no other knowledge that would require *Ann* and *Mary* to be the same. Thus, *Peter* is married to two different people, so the constraint (50) is not satisfied in $\mathcal{A}$.

5.2 Typing Constraints

Constraints can be used to check whether objects are correctly typed. Domain and range restrictions are typical forms of such constraints: for a role R and a concept C, they state that R-links can only point from or to objects that are explicitly typed as C. In this way, these constraints act as checks, saying that R-relationships can be asserted only for objects in C. The general form of domain constraints is

$$\exists R.\top \sqsubseteq C \tag{54}$$

whereas for range constraints it is

$$\top \sqsubseteq \forall R.C. \tag{55}$$

A typical example of a domain constraint is (9), which ensures that a name can be given only to objects that are either *bioSource*, *entity*, or *dataSource*. Another example is the following axiom, which states that it is only possible to be married to a *Person*:

$$\top \sqsubseteq \forall marriedTo.Person \tag{56}$$

To understand the difference in treating (56) as a standard axiom or a constraint, consider an ABox $\mathcal{A}$ containing only the fact (52). If (56) were a part of the standard TBox $\mathcal{S}$, then $\mathcal{A} \cup \mathcal{S}$ would be satisfiable; furthermore, due to (56), we would derive $Person(Ann)$. If we put (56) into the constraint TBox $\mathcal{C}$, then the only minimal model of $\mathcal{A}$ contains only the fact (52). Thus, *Ann* is not explicitly typed to be a *Person*, so the constraint (56) is not satisfied in $\mathcal{A}$.

5.3 Restrictions to Known Individuals

Sometimes, we might want to check whether certain objects are known by name. For example, an application for the management of tax returns might deal with two types of people: those who have submitted a tax return for processing, and those who are somehow related to the people from the first group (e.g., their spouses or children). For the application to function properly, it might not be necessary to explicitly specify the SSN for all people; only the SSNs for the people from the first group are of importance. Hence, in such an application we might use axioms (1)–(3) not as constraints, but as part of the standard TBox $\mathcal{S}$. Furthermore, to distinguish people who have submitted the tax return, we would introduce a concept *PersonTR* for persons with a tax return, and would make it a subset of *Person* in $\mathcal{S}$:

$$PersonTR \sqsubseteq Person \tag{57}$$

Two things should hold for each instance of *PersonTR*: first, we might require each such person to be explicitly known by name, and second, we might require the SSN of each such person to be known by name as well. Although constraints can be used to check whether an individual is present in an interpretation, they cannot distinguish named (known) from unnamed (unknown) individuals. We can, however, solve this problem using the following "trick." We can use a special concept O to denote all individuals known by name and state the following two constraints:

$$PersonTR \sqsubseteq O \tag{58}$$

$$PersonTR \sqsubseteq \exists hasSSN.(O \sqcap SSN) \tag{59}$$

Furthermore, we add the following ABox assertion for each individual a occurring in an ABox:

$$O(a) \tag{60}$$

Now in any minimal model of $\mathcal{S} \cup \mathcal{A}$, the assertions of the form (60) ensure that O is interpreted exactly as the set of all known objects. Hence, (58) ensures that each *PersonTR* is known, and (59) ensures that the social security number for each person is known as well.

One can object that this solution is not completely model-theoretic: it requires asserting (60) for each known individual, which is a form of procedural preprocessing. We agree that our solution is not completely clean in that sense; however, we believe that it is simple to understand and implement and thus acceptable.

For TBox reasoning, statements of the form (60) are, by definition, not taken into account. Finally, instead of the axioms (60), one could be tempted to use the following statement, where a_i are all individuals from the ABox:

$$O \equiv \{a_1, \ldots, a_n\} \tag{61}$$

This, however, requires nominals in the DL language, which makes reasoning more difficult [25]. Furthermore, since O occurs only in the constraint axioms, the axioms of the form (60) are sufficient: the minimal model semantics ensures that O contains exactly the individuals $a_1, \ldots, a_n$.

6. CONSTRAINT CHECKING

We now briefly discuss the algorithms for checking constraint satisfaction; since the semantics of all other reasoning problems is defined as usual, the existing algorithms can be applied to solve them. Due to space limitations, we present here only the intuition; for details, please refer to [19].

The complexity of constraint checking is primarily determined by the complexity of the standard TBox $\mathcal{S}$. To pinpoint the main source of difficulty, we use the following definition: a standard TBox $\mathcal{S}$ is *existential-free* if its axioms contain neither existential quantifiers under positive nor universal quantifiers under negative polarity. For example, such an $\mathcal{S}$ is allowed to contain an axiom of the form

$$\exists R.C \sqsubseteq D \tag{62}$$

but not an axiom of the form

$$C \sqsubseteq \exists R.D. \tag{63}$$

The axioms of existential-free TBoxes cannot imply the existence of unknown individuals. For example, the axiom (62) concerns only cases where there already is an R-link to an object in C, whereas, for each object in C, the axiom (63) implies the existence of an individual connected through an R-link to some individual in D. Therefore, reasoning with an existential-free TBox $\mathcal{S}$ is easier since we can limit our attention to the individuals explicitly mentioned in $\mathcal{A} \cup \mathcal{S}$. For finite ontologies, the number of such individuals is finite, so we only need to consider finitely many finite models, which implies that constraint checking is decidable.

Knowledge bases that are not existential-free have, in contrast, infinite models, and the number of such models can be infinite. We can, however, restrict our attention to infinite *forest-like* models, which consist of infinite trees possibly connected at their roots. To obtain a decision procedure, we clearly cannot examine such structures in their entirety. In [19], we present a decision procedure for checking constraint satisfaction if $\mathcal{S}$ is an $\mathcal{ALCHI}$ knowledge base. The algorithm is based on a reduction of our problem to the problem of checking satisfiability of an *SkS* formula—a monadic second-order formula over k-ary infinite trees [22]. Intuitively, we encode $\mathcal{K}$ in an *SkS* formula φ such that the forest-like models of $\mathcal{K}$ correspond to the models of φ. To enforce minimality of models, we use the second-order quantification of *SkS*. Due to its high computational complexity, we do not expect this algorithm to be used in practice; rather, it should be taken as evidence that constraint satisfaction checking is possible for nontrivial logics. In future, we shall try to establish tight complexity bounds and derive a more practical algorithm.

In the rest of this section, we focus on existential-free knowledge bases. For such knowledge bases, we can check constraint satisfaction using the well-known methods of logic programming. Given $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$, we proceed as follows.

1. We translate the standard TBox $\mathcal{S}$ into a first-order formula $\pi(\mathcal{S})$ according to the standard DL semantics [3] and translate the result into a (possibly disjunctive) logic program $\mathsf{LP}(\mathcal{S})$. $\mathsf{LP}(\mathcal{S})$ can be exponentially larger than $\mathcal{S}$; we address this drawback in [19].

2. For each rule in $\mathsf{LP}(\mathcal{S})$ in which a variable x occurs in the head but not in the body, we add to the rule body the literal $HU(x)$. For each individual a occurring in $\mathcal{A} \cup \mathcal{S}$, we introduce an assertion $HU(a)$.

3. We translate the constraint TBox $\mathcal{C}$ into a first-order formula $\varphi = \pi(\mathcal{C})$.

4. With each subbormula of φ, we associate a unique predicate E_φ. Then, for $\mu(\varphi)$ and $\mathsf{sub}(\varphi)$ as defined in Table 1, we compute the following logic program:

$$\mathsf{CN}(\mathcal{C}) = \mu(\varphi) \cup \bigcup_{\psi \in \mathsf{sub}(\psi)} \mathsf{CN}(\psi)$$

5. Then, $\mathcal{K}$ satisfies the constraint TBox $\mathcal{C}$ if and only if $\mathcal{A} \cup \mathsf{LP}(\mathcal{S}) \cup \mathsf{CN}(\mathcal{C}) \models_c E_\varphi$, where $\models_c$ denotes the well-known entailment in stratified (possibly disjunctive) logic programs.

The transformation in Step 4 of the algorithm is similar to the well-known Lloyd-Topor transformation of first-order queries in bodies of logic programs [16].

The intuition behind this algorithm is as follows. The minimal models of the positive logic program $\mathcal{A} \cup \mathsf{LP}(\mathcal{S})$ are exactly the same as the models of $\mathcal{A} \cup \mathsf{sk}(\pi(\mathcal{S}))$. Furthermore, $\mathsf{CN}(\mathcal{C})$ simply evaluates $\mathcal{C}$ and ensures that E_φ holds in a model I if and only if $\mathcal{C}$ is true in I. Thus, we derive E_φ if and only if $\mathcal{C}$ is satisfied in all minimal models.

Clearly, the program $\mathsf{CN}(\mathcal{C})$ is stratified, so the complexity of constraint checking is determined by the program $\mathsf{LP}(\mathcal{S})$. If the latter is nondisjunctive, then constraint checking and, by Theorem 4.2, answering positive queries can be performed in polynomial time; this holds even if $\mathcal{S} \cup \mathcal{C}$ is expressed in a DL such as $\mathcal{SHOIN}(\mathbf{D})$. Thus, our approach allows for efficient query answering for languages beyond existing polynomial DLs (e.g., DL-lite [9] or $\mathcal{EL}++$ [2]), provided that the "difficult" axioms are treated as constraints.

We apply this algorithm to $\mathcal{S}_{10}$, $\mathcal{C}_{10}$, and $\mathcal{A}_{10}$ from Section 4. The program $\mathsf{CN}(\mathcal{S}_{10})$ consists of the following rules:

$$Pet(x) \leftarrow Cat(x) \tag{64}$$

$$PetOwner(x) \leftarrow hasPet(x, y) \wedge Pet(y) \tag{65}$$

Furthermore, $\mathsf{CN}(\mathcal{C}_{10})$ consists of the following rules:

$$E_1(x) \leftarrow hasPet(x, y) \wedge Cat(y) \tag{66}$$

$$E_2 \leftarrow CatOwner(x), not\ E_{\exists hasPet.Cat}(x) \tag{67}$$

$$E_\varphi \leftarrow not\ E_2 \tag{68}$$

Now $\mathcal{A}_{10} \cup \mathcal{S}_{10} \models_c E_\varphi$, so the constraints are satisfied.

7. CONCLUSION

Motivated by the problems encountered in the applications of OWL to data-centric problems, we have compared OWL and relational databases w.r.t. their approaches to schema modeling, schema and data reasoning problems, and constraint checking. We have seen that reasoning about the schema in databases and DLs is, in both cases, performed under standard first-order semantics, and it even employs closely related algorithms. The differences between relational databases and OWL become apparent when we consider the problems related to reasoning about data. In relational databases, answering queries and constraint satisfaction checking correspond to model checking, whereas, in DLs, the only form of constraint checking available is checking satisfiability of an ABox w.r.t. a TBox—a problem that is not concerned with the form of the data. This often causes misunderstandings in practice: OWL ontologies can be understood as incomplete databases, while the databases encountered in practice are often complete.

To control the degree of incompleteness in an ontology, we have proposed the notion of *extended* DL knowledge bases, in which a certain subset of TBox axioms can be designated as constraints. For TBox reasoning, constraints behave like normal TBox axioms; for ABox reasoning, however, they are interpreted in the spirit of relational databases. That is, we define the semantics of constraint satisfaction in such a way that it indeed checks whether all mandatory assertions are entailed by the given ABox and TBox.

We have also shown that, if the constraints are satisfied, we can disregard them while answering positive queries. This indicates that our semantics of constraint satisfaction is indeed reasonable, and means that answering queries under constraints may be computationally easier since we may have a smaller input TBox. Finally, we have sketched algorithms for checking constraint satisfaction.

Table 1: Translating Formulae into Constraints

	φ	$\mu(\varphi)$	$\mathsf{sub}(\varphi)$
1	$A(t_1, \ldots, t_m)$	$A(t_1, \ldots, t_m) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\emptyset$
2	$\neg\psi$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge not\ E_\psi(x_1, \ldots, x_n) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi\}$
3	$\psi_1 \wedge \psi_2$	$E_{\psi_1}(y_1, \ldots, y_m) \wedge E_{\psi_2}(z_1, \ldots, z_k) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi_1, \psi_2\}$
4	$\psi_1 \vee \psi_2$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge E_{\psi_1}(y_1, \ldots, y_m) \rightarrow E_\varphi(x_1, \ldots, x_n)$ $HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge E_{\psi_2}(z_1, \ldots, z_k) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi_1, \psi_2\}$
5	$\exists y : \psi$	$E_\psi(y_1, \ldots, y_m) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi\}$
6	$\forall y : \psi$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge not\ E_{\exists y:\neg\psi}(x_1, \ldots, x_n) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi\}$
7	$\exists^{\geq k} y : \psi$	$\bigwedge_{i=1}^{k} E_\psi(y_1, \ldots, y_m)[y^i/y] \wedge \bigwedge_{1 \leq i < j \leq k} not\ y^i \approx y^j \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi\}$
8	$\exists^{\leq k} y : \psi$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge not\ E_{\exists \geq k+1 y:\neg\psi}(x_1, \ldots, x_n) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\psi\}$

Note: $x_1, \ldots, x_n$ are the free variables of φ; $y_1, \ldots, y_m$ are the free variables of ψ and ψ_1; and $z_1, \ldots, z_k$ are the free variables of ψ_2. The predicate A can be $\approx$.

In future, we will try to obtain tight complexity bounds for constraint checking in the second case. Furthermore, we plan to implement our approach in the OWL reasoner KAON2 and test its usefulness on practical problems.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. IJCAI 2005*, pages 364–369, Edinburgh, UK, 2005.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proc. IJCAI '91*, pages 452–457, Sydney, Australia, 1991.

[5] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5–40, 2001.

[6] P. Bonatti, C. Lutz, and F. Wolter. Description Logics with Circumscription. In *Proc. KR 2006*, pages 400–410, Lake District, UK, 2006.

[7] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[8] D. Calvanese, D. D. Giacomo, and M. Lenzerini. Keys for free in description logics. In *Proc. DL 2000*, Aachen, Germany, 2000.

[9] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data Complexity of Query Answering in Description Logics. In *Proc. KR 2006*, pages 260–270, Lake District, UK, 2006.

[10] D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. PODS '98*, pages 149–158, Seattle, WA, USA, 1998.

[11] F. M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.

[12] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. In *Proc. IJCAI 2007*, Hyderabad, India, 2007.

[13] V. Haarslev and R. Möller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proc. DL 2003*, Rome, Italy, 2003.

[14] S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Conceptual Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 2006. To appear.

[15] A. Y. Levy. Obtaining Complete Answers from Incomplete Databases. In *Proc. VLDB '96*, pages 402–412, Mumbai, India, 1996.

[16] J. W. Lloyd and R. W. Topor. Making Prolog More Expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.

[17] C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, Nominals, and Concrete Domains. *Journal of Artificial Intelligence Research*, 23:667–726, 2005.

[18] C. Lutz, U. Sattler, and L. Tendera. The Complexity of Finite Model Reasoning in Description Logics. *Information and Computation*, 199:132–171, 2005.

[19] B. Motik, I. Horrocks, and U. Sattler. Integrating Description Logics and Relational Databases. Technical report, University of Manchester, UK, 2006.

[20] B. Motik and R. Rosati. A Faithful Integration of Description Logics with Logic Programming. In *Proc. IJCAI 2007*, Hyderabad, India, 2007.

[21] A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science, 2001.

[22] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transations of the American Mathematical Society*, 141:1–35, 1969.

[23] R. Reiter. What Should a Database Know? *Journal of Logic Programming*, 14(1–2):127–153, 1992.

[24] R. Rosati. $\mathcal{DL}+log$: A Tight Integration of Description Logics and Disjunctive Datalog. In *Proc. KR 2006*, pages 68–78, Lake District, UK, 2006.

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