Integrating Description Logics and Relational Databases

Boris Motik, Ian Horrocks, Ulrike Sattler

Department of Computer Science
University of Manchester
Manchester, UK

December 6, 2006

Abstract

In this paper, we compare description logics with relational databases with respect to their treatment of schema constraints, the languages used to express these constraints, and the approaches to query answering and constraint checking. Our analysis reveals a significant overlap between the two formalisms. Inspired by the integrity constraints of relational databases, we define a notion of integrity constraints for description logics. We analyze different possibilities for defining the semantics of the constraints. Finally, we present several algorithms for checking constraint satisfaction for description logics with varying degrees of expressivity.

Contents

1 Introduction

Description Logics (DLs) [3] are a family of knowledge representation formalisms with a well-defined model-theoretic semantics and well-understood computational properties. DLs were successfully applied to many problems in computer science; furthermore, they form the logical underpinning of the Web Ontology Language (OWL)—a language for ontology modeling standardized by the World Wide Web Consortium (W3C). DLs and OWL provide numerous constructs for expressing very complex domain models, so they were often used for modeling schemas in data management applications. Experience in building such applications has, however, revealed a gap between DLs and typical use cases; this gap becomes more noticeable if DLs are compared to standard relational databases.

To understand these problems, consider the following example. In an application for managing tax returns, one might require each person to have a social security number. In a relational database, this would be typically captured by an inclusion dependency stating that, for each person, a social security number must exist. To ensure data integrity, such a dependency would be interpreted as an integrity constraint during database updates: whenever a person is added to a database, the database system would check whether that person's social security number has been specified as well, and if not, the update would be rejected. In a DL knowledge base, the same requirement would typically be captured by an axiom asserting that every person must have a social security number. Adding a person without a social security number to the resulting knowledge base would not, however, be flagged as an error; rather, that person would be inferred to have some unknown social security number.

There is a long research tradition in extending logic-based knowledge representation formalisms with database-like integrity constraints. In his seminal paper, Reiter observed that integrity constraints are not objective sentences about the world; rather, they describe the state of the database, and are therefore of an epistemic nature [36]. Hence, most extensions of DLs with integrity constraints are based on autoepistemic extensions of DLs, such as the description logics of minimal knowledge and negation-as-failure [15] or various nonmonotonic rule extensions of DLs [37, 31].

While these approaches do solve the problem to a certain extent, the solution is not in the spirit of relational databases. As we discuss in more detail in Section 7, the constraints in these approaches do not affect TBox reasoning at all; they are only applied to ABox individuals. Such constraints are thus very weak, as they do not say anything about the structure of the world; they only constrain the structure of ABoxes. Furthermore, the same statement can usually be expressed as either an ordinary axiom or as a constraint. While these two forms are naturally related, there is almost no

semantic relationship between them. Therefore, choosing an appropriate way to model a statement requires significant cognitive effort.

In relational databases, however, integrity constraints have a dual role: on the one hand, they describe all possible worlds, and, on the other hand, they describe the allowed states of the database [1]. Integrity constraints are used in data reasoning tasks, such as checking the integrity of a database, as well as in schema reasoning tasks, such as computing query subsumption. The semantic relationship between these two roles of constraints is much clearer, which significantly simplifies modeling.

To clarify these issues, in Section 3, we analyze the relationship between the schema constraints in DLs and relational databases. Then, in Section 4, we propose the notion of *extended DL knowledge bases*. Such knowledge bases allow the modeler to designate a subset of TBox axioms as constraints that have the usual semantics for TBox reasoning, but are interpreted as checks during ABox reasoning. We also show that, if an ABox satisfies the constraints, we can disregard the constraints while answering a broad class of ABox queries. Our approach thus improves the performance of query answering as it reduces the set of relevant TBox axioms.

In Section 5, we discuss the typical usage patterns for our approach. Furthermore, in Section 6, we present several algorithms for checking constraint satisfaction for different types of knowledge bases. Finally, in Section 7, we discuss how our approach relates to the existing approaches based on autoepistemic extensions of DLs.

2 Preliminaries

2.1 Description Logics

The approach presented in this paper can be applied to many different description logics, so we present here only a high-level overview of the syntax and the semantics of description logics without going into details. For a formal definition, please refer to [3].

The building blocks of DL knowledge bases are *concepts* (or *classes*), representing sets of objects, *roles* (or *properties*), representing relationships between objects, and *individuals*, representing specific objects. Concepts such as *Person* are called *atomic*. Using concept constructors, one can construct *complex* concepts; for example, the concept $\exists \mathit{hasFather}.\mathit{Person}$ describes those objects that are related through the *hasFather* role with an object from the concept *Person*. Expressive DLs provide a rich set of concept constructors, such as the Boolean connectives, existential and universal quantification, and number restrictions.

A DL knowledge base $\mathcal{K}$ typically consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. A TBox contains axioms about the general structure of all allowed worlds, and is therefore in its purpose akin to a database schema. For example,

the TBox axiom (1) states that each instance of the concept *Person* must be related by the role *hasFather* with an instance of the concept *Person*. An ABox contains axioms describing the structure of particular worlds. For example, the ABox axiom (2) states that *Peter* is a *Person*, and (3) states that *Paul* is a brother of *Peter*.

$$Person \sqsubseteq \exists hasFather.Person \tag{1}$$

$$Person(Peter) \tag{2}$$

$$hasBrother(Peter, Paul) \tag{3}$$

A DL knowledge base $\mathcal{K}$ can be given semantics either directly, or by translating it into a formula $\pi(\mathcal{K})$ of first-order logic with equality. Atomic concepts are translated into unary predicates, complex concepts are translated into formulae with one free variable, and roles are translated into binary predicates. For example, the axiom (1) can be represented as the following first-order formula:

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

Given a knowledge base $\mathcal{K}$, two basic DL reasoning problems are checking if an individual a is an instance of a concept C (written $\mathcal{K} \models C(a)$) or if a concept C is subsumed by another concept D (written $\mathcal{K} \models C \sqsubseteq D$).

2.2 Disjunctive Logic Programs

We now review the syntax and the semantics of disjunctive logic programs [18, 19]. An atom is a formula of the form $A(t_1, \ldots, t_n)$ or $t_1 \approx t_2$, where t_i are first-order terms (we allow the terms to contain function symbols). A *disjunctive logic program with equality* P is a finite set of *rules* of the following form, where A_i are *head atoms*, B_i^+ are *positive body atoms*, and *not* B_i^- are *negative body atoms*:

$$A_1 \vee \ldots \vee A_n \leftarrow B_1^+ \wedge \ldots \wedge B_m^+ \wedge not\, B_1^- \wedge \ldots \wedge not\, B_k^-$$

A *fact* is a rule with $m = k = 0$. A rule is *safe* if each variable occurring in the rule also occurs in some positive body atom B_i^+. Typically, each program P is allowed to contain only safe rules.

The Herbrand base of a program P is a set of all ground terms over the signature of P (we assume that P contains at least one constant). The grounding of P, written $\mathsf{gr}(P)$, is the ground (possibly infinite) set of rules obtained by replacing in each $r \in P$ all variables with all terms from the Herbrand base in all possible ways. An *interpretation* I of a program P is a set of positive ground atoms formed over the Herbrand base of P in which $\approx$ is interpreted as a congruence relation [17]. An interpretation I satisfies a positive atom A, written $I \models A$, if and only if $A \in I$, and I satisfies a

negative atom *not* A, written $I \models \mathit{not}\, A$, if and only if $A \notin I$. Furthermore, I satisfies a ground rule r, written $I \models r$, if either some head atom of r is satisfied in r or some body atom is not satisfied in I; finally, I satisfies a nonground program P if $I \models r$ for each $r \in \mathsf{gr}(P)$. In the latter case, I is a *model* of P. If I is a model of P but each $I' \subset I$ is not a model of P, then I is a *minimal model* of P. For a model I, the Gelfond-Lifshitz reduct P^I is the set of rules obtained from $\mathsf{gr}(P)$ by (i) removing all rules containing a negative atom $\mathit{not}\, B_i^-$ such that $I \not\models \mathit{not}\, B_i^-$, and (ii) removing all negative body atoms from the remaining rules. An interpretation I is a *stable model* of P if I is a minimal model of P^I. A program P *cautiously entails* a (positive or negative) ground atom α, written $P \models_c \alpha$, if $I \models \alpha$ for each stable model I of P.

Conventional definitions of logic programs do not allow for explicit equality (at least not in the rule heads). In our approach, we treat $\approx$ as a special predicate interpreted as a congruence. In fact, $\approx$ could be just an ordinary predicate for which the standard properties of equality have been explicitly axiomatized [17]. Note that the extension of $\approx$ is subjected to minimization in the definition of the semantics, just like any other predicate.

3 A Comparison of DLs and Relational Databases

We now compare in detail various aspects of description logics and relational databases.

3.1 Schema Language

The schema of a DL knowledge base is typically expressed as a TBox (terminology box), which is just a set of axioms. For example, one might state that each person must have a social security number (SSN); furthermore, any person can have at most one SSN and, conversely, each SSN can be assigned to at most one individual. These three statements are expressed using the following TBox axioms:

$$\mathit{Person} \sqsubseteq \exists \mathit{hasSSN}.\mathit{SSN} \tag{5}$$

$$\mathit{Person} \sqsubseteq \leq 1\, \mathit{hasSSN} \tag{6}$$

$$\mathit{SSN} \sqsubseteq \leq 1\, \mathit{hasSSN}^- \tag{7}$$

Most DLs are just decidable fragments of first-order logic, so their axioms have an equivalent representation as first-order formulae, cf. [9]. For example, the axioms (5)–(7) are translated into first-order logic as follows:

$$\forall x : [\mathit{Person}(x) \rightarrow \exists y : \mathit{hasSSN}(x, y) \wedge \mathit{SSN}(y)] \tag{8}$$

$$\forall x, y_1, y_2 : [\mathit{Person}(x) \wedge \mathit{hasSSN}(x, y_1) \wedge \mathit{hasSSN}(x, y_2) \rightarrow y_1 \approx y_2] \tag{9}$$

$$\forall x, y_1, y_2 : [\mathit{SSN}(x) \wedge \mathit{hasSSN}(y_1, x) \wedge \mathit{hasSSN}(y_2, x) \rightarrow y_1 \approx y_2] \tag{10}$$

The schema of relational databases is defined in terms of relations and dependencies among them. Different types of dependencies were considered in literature, such as functional, inclusion, or join dependencies. As discussed in [1], most types of dependencies can be represented as first-order formulae of the following form, 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:

$$(11) \quad \forall x_1, \ldots, x_n : [\psi(x_1, \ldots, x_n) \rightarrow \exists y_1, \ldots, y_m : \xi(x_1, \ldots, x_n, y_1, \ldots, y_m)]$$

Although the expressivity of DLs and of relational dependencies is clearly different, the underlying principles between DLs and relational databases are quite closely related. In fact, the formula (8) corresponds to an inclusion dependency, whereas the formulae (9) and (10) correspond to key dependencies.

3.2 Interpreting the Schema

DL TBoxes and relational schemas are interpreted according to standard first-order semantics, in the sense that they describe the properties of the allowed relational structures. In DLs, these structures are called *models*, whereas in relational databases they are called *database instances*; the underlying principle is, however, almost the same.

There is a slight technical difference between models and database instances: models are usually allowed to be infinite, whereas database instances are typically required to be finite. The restriction to finite database instances 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 expressive dependency or DLs languages, this does not necessarily hold; for example, there are DL knowledge bases that are satisfied only in infinite models [3]. Even though the complexity of finite model reasoning is, for many DLs, the same as the complexity of reasoning w.r.t. arbitrary models, the former is usually more involved [30]. Therefore, in the rest of this paper, we drop the restriction to finite database instances and consider models and database instances to be synonyms.

3.3 Domains and Typing

Relational databases sometimes assign types to relation columns; for example, one might say that the second position of *hasSSN* must be a string of a certain form. This is quite important in practice since the physical layout of a database is derived from the column types. In contrast, column types are not considered in many theoretical works (e.g., in algorithms for checking

containment of conjunctive queries); rather, it is assumed that each column draws its values from a common countable domain set [1].

Most DLs do not type their predicates, but interpret them over an arbitrary *abstract domain* set. Many practical applications, however, need the capability to represent concrete data. These use-cases can often be addressed by extending the DLs with so-called *concrete domains* along the principles first outlined in [4].

In this paper, we consider neither typed relational schemas nor DL knowledge bases with concrete domains, and simply interpret both relational schemata and DL TBoxes in first-order logic. This simplifies both formalisms significantly. For example, adding key constraints to untyped DLs is not a problem [11], whereas adding them to DLs with typed predicates is much more involved [29].

3.4 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 a DL TBox, the entailed subsumption relationships can be used to detect modeling errors. Furthermore, concept subsumption has been used to optimize query answering [21], especially when generalized to subsumption of conjunctive queries [13, 23, 20]. Another important TBox inference is checking concept satisfiability—that is, determining whether a model of $\mathcal{T}$ exists in which the concept has a nonempty extension. If concepts are unsatisfiable, this is usually due to modeling errors, so this inference is also quite useful in ontology modeling.

Reasoning about the schema is certainly not the most prominent feature of relational database systems, so it may come as a surprise that a significant amount of database research has been devoted to it. The most important inference is checking containment of conjunctive queries [1]: a query Q_1 is contained in a query Q_2 with respect to 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, where complex queries are replaced with equivalent, but simpler queries that can be answered more efficiently. Another inference often considered in the literature is dependency minimization—that is, computing an equivalent schema that is minimal with respect to the given one.

In both DLs and relational databases, schema reasoning problems correspond to checking whether some formula φ holds in each model (database instance) that satisfies the schema $\mathcal{T}$—that is, to checking whether $\mathcal{T} \models \varphi$. In other words, the terminological problems in both DLs and relational databases involve checking 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 [6], whereas the state-of-the-art reasoning technique in relational database is chase [1]. Apart from notational differences, the principles underlying these two techniques are the same: they both use a set of rules to construct a model that satisfies the schema $\mathcal{T}$ but not the formula φ.

To summarize, from a semantic point of view, both DLs and databases treat schema reasoning problems in the same way. Therefore, DLs can be understood as very 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.

3.5 Query Answering

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 that satisfies 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 [23, 24]. A query in a DL system can be viewed as a first-order formula φ with the free variables $x_1, \ldots, x_n$. Just like the schema reasoning problems, the query answering problems in description logics are defined as first-order entailment (i.e., they consider all possible models of $\mathcal{K}$): a tuple $a_1, \ldots, a_n$ is an 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 schema $\mathcal{K}$; 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 [27]—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 constraints specifying how the incomplete extensions relate to the actual (unknown) database instance.

Queries in incomplete databases are also (possibly syntactically restricted) first-order formulae. In contrast to complete databases, a tuple $a_1, \ldots, a_n$ is an 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.

3.6 Constraint Checking

In description logics, one can check whether an ABox $\mathcal{A}$ is consistent with a TBox $\mathcal{T}$—that is, whether a model I satisfying 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 (5)–(7), and let $\mathcal{A}$ contain only the following axiom:

$$Person(Peter) \tag{12}$$

If (5) 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 (5) is not interpreted as a check; rather, it implies the existence of some (unknown) SSN. The axioms in $\mathcal{T}$ describe the allowed models, but they do not restrict the allowed ABoxes. In fact, DLs do not provide for database-like integrity constraints at all.

In contrast, constraints play a central role in relational databases, where they are used to ensure they integrity of data. As in DLs, a relational schema $\mathcal{T}$ is a set of formulae that must hold for any database instance. In contrast to DLs, relational databases consider only one instance at a time; therefore, constraint checking is, just like query answering, interpreted as model checking. Hence, given a database instance I, a relational database checks the satisfaction of the schema constraints $\mathcal{T}$ by checking whether $I \models \mathcal{T}$. (In practice, constraints are incrementally checked after database updates; these dynamic aspects are, however, not important for this discussion.) 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.

3.7 Discussion

Users of DL systems often complain about the "open-world semantics of DLs" or "the unintuitive semantics of constraints." If accepted at face value, these complaints can be quite misleading because they actually describe the symptoms, and not the causes of a serious misunderstanding.

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 any acceptable database instance. When computing the subsumption relationship between concepts or queries, we do not have a fixed instance at all. Therefore, the closed-world assumption in the traditional sense is meaningless: a TBox $\mathcal{T}$ (with no assertions about individuals) does not imply any ground facts, so, under the standard closed-world assumption, we have $\mathcal{T} \models \neg\alpha$ for each ground atom α. Hence, to draw useful consequences, we must interpret $\mathcal{T}$ under open-world semantics.

It is possible to define a notion of query subsumption for nonmonotonic formalisms that employ a variant of closed-world semantics. Such problems are, however, typically highly undecidable. Even for datalog without negation—one of the simplest nonmonotonic formalisms—query equivalence is undecidable [38]. Therefore, first-order semantics is a more appropriate choice if decidability and practicability are desired.

The gap between DLs and users' expectations is much larger regarding the approach to data management. 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 the first-order logic). In contrast, DL ABoxes actually correspond to incomplete (relational) databases.

Our experience with building DL applications has shown that information about individuals is often complete. This is particularly true in data management applications, whose primary goal is to manage large amounts of data structured according to relatively simple schemata. In such applications, the mismatch between the features of DLs and the actual use-cases becomes quite apparent.

To understand the types of problem that occur in practice, consider the following example. Biopax[1] is an ontology developed by the bioinformatics community with the goal of facilitating data integration and interchange between biological databases. This ontology defines a role *NAME* and states its domain to be the union of *bioSource*, *entity*, and *dataSource*:

$$\exists NAME.\top \sqsubseteq bioSource \sqcup entity \sqcup dataSource \tag{13}$$

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 satisfies this constraint: each object that is named is also typed (sometimes indirectly through the class hierarchy) to at least one of the required classes. The axiom (13), 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, (13) cannot be used to check whether all data

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

is correctly typed. Furthermore, since the concept in the consequent is a disjunction, the axiom (13) requires reasoning by case, which are one of the reasons for intractability of DL reasoning algorithms [3, Chapter 3]. Hence, the axiom (13) causes two types of problems: on the one hand, it does not have the desired semantics, so we cannot check whether the data has been completely and correctly typed; on the other hand, it introduces a significant performance penalty for reasoning.

Even though some applications deal mostly with complete data, many applications require the representation of and reasoning about incomplete information. For example, we might have a rule that married people are eligible for a tax cut:

$$\exists \mathit{marriedTo}.\top \sqsubseteq \mathit{TaxCut} \tag{14}$$

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:

$$\exists \mathit{marriedTo}.\mathit{Woman}(\mathit{Peter}) \tag{15}$$

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.

In order to satisfy both of the above requirements, 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. Furthermore, the modeler should be able to check whether all data has been specified as required by the ontology. Finally, one should not pay an unnecessary performance penalty for using an expressive schema language. In the following section, we introduce a notion of extended DL knowledge bases that addresses some of the issues raised in this section.

4 Constraints for Description Logics

In this section, we define a notion of constraints for DL knowledge bases that attempts to mimic the integrity constraints of relational databases described in Section 3.

4.1 The Syntax and the Semantics

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 3 to DLs. 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:

$$Student(Peter) \tag{16}$$

$$hasSSN(Peter, nr12345) \tag{17}$$

$$SSN(nr12345) \tag{18}$$

$$Student(Paul) \tag{19}$$

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

$$Student \sqsubseteq Person \tag{20}$$

$$Person \sqsubseteq \exists hasSSN.SSN \tag{21}$$

Let us assume that we want to treat (21) as a constraint, but (20) as a normal axiom. Then, we derive $Person(Peter)$ and $Person(Paul)$ by (20). The axiom (21) is satisfied for *Peter* due to (16), (17), and (18); however, an SSN has not been specified for *Paul*, so we would expect (21) 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.

Definition 4.1. *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.*

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 KBs. 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 (16), $\mathcal{S}_2 = \emptyset$, and let $\mathcal{C}_2$ contain only the axiom (21). The interpretation $I = \{\mathit{Student}(\mathit{Peter}), \mathit{Person}(\mathit{Peter})\}$ 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 $\mathit{Person}(\mathit{Peter})$ 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: $\mathit{Person}(\mathit{Peter})$ 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:

$$\mathit{Cat}(\mathit{ShereKahn}) \tag{22}$$

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

$$\mathit{Cat} \sqsubseteq \mathit{Tiger} \sqcup \mathit{Leopard} \tag{23}$$

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

$$\mathit{Tiger} \sqsubseteq \mathit{Carnivore} \tag{24}$$

$$\mathit{Leopard} \sqsubseteq \mathit{Carnivore} \tag{25}$$

Now neither $\mathit{Tiger}(\mathit{ShereKahn})$ nor $\mathit{Leopard}(\mathit{ShereKahn})$ 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

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

This answer does not satisfy our intuition: in each model of $\mathcal{A}_3 \cup \mathcal{S}_3$, either $\mathit{Tiger}(\mathit{ShereKahn})$ or $\mathit{Leopard}(\mathit{ShereKahn})$ holds; however, the fact $\mathit{Carnivore}(\mathit{ShereKahn})$ does not necessarily hold in either case; hence, by treating (24)–(25) as constraints and not as standard axioms, we neither get a constraint violation nor derive the consequence $\mathit{Carnivore}(\mathit{ShereKahn})$.

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 (21) 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:

$$Cat \sqsubseteq (Tiger \sqcap Carnivore) \sqcup (Leopard \sqcap Carnivore) \tag{26}$$

Now the fact $Carnivore(ShereKahn)$ 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 (24)–(25) 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 [8] and in the semantics of open answer set programs [22]. Consider, however, the following ABox $\mathcal{A}_5$:

$$Woman(Alice) \tag{27}$$

$$Man(Bob) \tag{28}$$

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

$$Woman \sqcap Man \sqsubseteq \bot \tag{29}$$

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 $I' = \{ \mathit{Woman}(\mathit{Alice}), \mathit{Man}(\mathit{Bob}) \}$ (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:

$$\mathit{HasChild}(\mathit{Peter}) \tag{30}$$

$$\mathit{HasHappyChild}(\mathit{Peter}) \tag{31}$$

$$\mathit{TwoChildren}(\mathit{Peter}) \tag{32}$$

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

$$\mathit{HasChild} \sqsubseteq \exists \mathit{hasChild}.\mathit{Child} \tag{33}$$

$$\mathit{HasHappyChild} \sqsubseteq \exists \mathit{hasChild}.(\mathit{Child} \sqcap \mathit{Happy}) \tag{34}$$

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

$$\mathit{TwoChildren} \sqsubseteq \geq 2\, \mathit{hasChild}.\mathit{Child} \tag{35}$$

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 (33) and (34)—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:

$$\mathit{HasChild}(\mathit{Peter}) \tag{36}$$

$$\mathit{TwoChildren}(\mathit{Peter}) \tag{37}$$

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

$$\mathit{HasChild} \sqsubseteq \exists \mathit{hasChild}.\mathit{Child} \sqcap \exists \mathit{hasChild}.\mathit{Child} \tag{38}$$

As in the previous example, $\mathcal{C}_7$ is satisfied in $\mathcal{A}_7 \cup \mathcal{S}_7$ since (38) introduces two (possibly identical) individuals in the extension of *Child*. Let $\mathcal{S}'_7$ be a standard TBox containing only the axiom (39):

$$\mathit{HasChild} \sqsubseteq \exists \mathit{hasChild}.\mathit{Child} \tag{39}$$

Now $\mathcal{C}_7$ is not satisfied in $\mathcal{A} \cup \mathcal{S}'_7$ since (39) 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 (38) requires the existence of only one individual. Recall, however, that $\mathcal{S}_6$ and $\mathcal{S}_7$ are quite closely related: the effect of (38) with respect to *Child* is the same as that of (33) and (34). Hence, if (38) should introduce only one individual, then (33) and (34) 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 (33) and (34) should introduce different individuals, whereas the existential quantifiers in (38) 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* [33], 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 in Section 6.3. The next definition uses outer skolemization, in which the existential quantifiers are replaced with functional symbols from outermost to the innermost one.

Definition 4.2. *Let φ be a first-order formula and $\mathsf{sk}(\varphi)$ the formula obtained by outer skolemization of φ [33]. 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, 9].

Definition 4.3. *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: Definition 4.3 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 4.3 closely follows our intuition on the examples presented thus far. Furthermore, in Section 4.3 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:

$$Vegetarian(Ian) \tag{40}$$

$$eats(Ian, soup) \tag{41}$$

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

$$Vegetarian \sqsubseteq \forall eats.\neg Meat \tag{42}$$

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 (40) and (41) is the only minimal Herbrand model of $\mathcal{A}_8$ and $I \models \mathcal{C}_8$. In fact, the axiom (42) is equivalent to the following axiom:

$$Vegetarian \sqcap \exists eats.Meat \sqsubseteq \bot \tag{43}$$

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:

$$NotMeat \equiv \neg Meat \tag{44}$$

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

$$\textit{Vegetarian} \sqsubseteq \forall \textit{eats}.\textit{NotMeat} \tag{45}$$

The constraint (44) 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 $\mathcal{A}_9$:

$$\begin{aligned} I_1 &= \{\textit{Vegetarian}(\textit{Ian}), \textit{eats}(\textit{Ian}, \textit{soup}), \textit{Meat}(\textit{soup})\} \\ I_2 &= \{\textit{Vegetarian}(\textit{Ian}), \textit{eats}(\textit{Ian}, \textit{soup}), \textit{NotMeat}(\textit{soup})\} \end{aligned}$$

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

4.2 Alternative Characterization via Logic Programming

In this section we develop an alternative characterization of the semantics of extended knowledge bases that is based on logic programming. This is interesting because it gives a slightly more procedural flavor to the notion of derivability, which is useful for clarifying the intuition behind our approach.

To obtain our characterization, we first show how to evaluate the axioms from $\mathcal{C}$ using a stratified datalog program. This is somewhat reminiscent of the approach from [10].

Definition 4.4. *For a first-order formula χ, let E_χ be an n-ary predicate symbol uniquely associated with χ, where n is the number of the free variables of χ. For a first-order formula φ, the* constraint program $\mathsf{CN}(\varphi)$ *is defined recursively as follows, for μ and* sub *as defined in Table 1:*

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

The following claim trivially follows from Definition 4.4:

Proposition 4.5. *The program $\mathsf{CN}(\varphi)$ is stratified and nonrecursive.*

We now show how we can use $\mathsf{CN}(\varphi)$ to evaluate a formula φ in a Herbrand model.

Lemma 4.6. *For a Herbrand model I, let $\mathcal{A}(I)$ be exactly the set of facts containing I and a fact $HU(t)$ for each ground term t from the universe of I. For a first-order formula φ with free variables $\mathbf{x}$ and a tuple of ground terms $\mathbf{t}$, we have $I \models \varphi[\mathbf{t}/\mathbf{x}]$ if and only if $\mathcal{A}(I) \cup \mathsf{CN}(\varphi) \models_c E_\varphi(\mathbf{t})$.*

Proof. The proof is by an easy induction on the structure of φ. For the induction base, if φ is an atomic formula, then $\mathsf{CN}(\varphi)$ contains a rule of the

Table 1: Translating Constraints into Rules

	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	$\forall y : \psi$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge not\, E_{\exists y:\neg\psi}(x_1, \ldots, x_n) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\exists y : \neg\psi\}$
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	$\exists^{\leq k} y : \psi$	$HU(x_1) \wedge \ldots \wedge HU(x_n) \wedge not\, E_{\exists^{\geq k+1} y:\neg\psi}(x_1, \ldots, x_n) \rightarrow E_\varphi(x_1, \ldots, x_n)$	$\{\exists^{\geq k+1} y : \neg\psi\}$

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 $\approx$.

form (1) from Table 1, and the claim is obvious. Let us now consider the possible forms of a complex formula φ.

For $\varphi = \neg\psi$, the program $\mathsf{CN}(\varphi)$ contains a rule of the form (2) from Table 1. This rule ensures that $E_\varphi(t_1, \ldots, t_n)$ holds exactly if $E_\psi(t_1, \ldots, t_n)$ does not hold, which implies the claim. The cases for $\varphi = \psi_1 \wedge \psi_2$ and $\varphi = \psi_1 \vee \psi_2$ can be proved straightforwardly in similar way.

For $\varphi = \exists y : \psi$, the program $\mathsf{CN}(\varphi)$ contains a rule of the form (5) from Table 1. This rule ensures that $E_\varphi(t_1, \ldots, t_n)$ holds whenever there is some ground term s such that $E_\psi(t_1, \ldots, s, \ldots, t_n)$ holds, which implies the claim.

For $\varphi = \forall y : \psi$, the program $\mathsf{CN}(\varphi)$ contains a rule of the form (6) which is based on the fact that φ is equivalent to $\varphi = \neg\exists y : \neg\psi$. Finally, for $\exists^{\geq k} y : \psi$ and $\exists^{\leq k} y : \psi$, the claim follows from the standard translation of counting quantifiers into first-order logic. □

We next define how to convert a first-order formula φ into an equivalent logic program $\mathsf{LP}(\varphi)$. By the following definition, $\mathsf{LP}(\varphi)$ can be exponentially large in the size of φ. At this point, we are interested only in the semantic properties of $\mathsf{LP}(\varphi)$ and address the exponential blowup in Section 6.1.

Definition 4.7. *For a first-order formula φ, let φ' be the translation of $\mathsf{sk}(\varphi)$ into conjunctive normal form, and let $\mathsf{LP}(\varphi)$ be the logic program obtained from φ' by*

- *converting each clause $\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$ into a rule $A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$;*
- *adding an atom $HU(x)$ to the body of each rule in which the variable x occurs in the head but not in the body;*
- *adding a fact $HU(c)$ for each constant c; and*
- *adding the following rule for each n-ary function symbol f:*

$$HU(x_1) \wedge \ldots \wedge HU(x_n) \rightarrow HU(f(x_1, \ldots, x_n))$$

We are now ready to embed constraint checking into logic programming. In the following discussion, we consider negative ABox axioms $\neg A(a)$ and $a \not\approx b$ as the rules $\leftarrow A(a)$ and $\leftarrow a \approx b$.

Theorem 4.8. *An extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ satisfies the constraints in $\mathcal{C}$ if and only if $\mathcal{A} \cup \mathsf{LP}(\varphi) \cup \mathsf{CN}(\psi) \models_c E_\psi$, where $\varphi = \pi(\mathcal{S})$ and $\psi = \pi(\mathcal{C})$.*

Proof. The formula φ' in Definition 4.7 is obtained from $\mathsf{sk}(\varphi)$ using standard equivalences of first-order logic, which preserve satisfiability of formulae in any model, so the minimal Herbrand models of $\mathsf{sk}(\mathcal{S} \cup \mathcal{A})$ and $\{\varphi'\} \cup \mathcal{A}$ coincide. Furthermore, the facts and the rules introduced in the third and

the fourth item of Definition 4.7 just enumerate the entire Herbrand universe, so each minimal Herbrand model of $\{\varphi'\} \cup \mathcal{A}$ corresponds exactly to one minimal Herbrand model of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A}$ augmented with $HU(t)$ for each ground term t. The program $\mathsf{CN}(\mathcal{C})$ is stratified and its rules contain only the predicates E_χ in their heads, so $\mathsf{CN}(\mathcal{C})$ just extends each minimal model I of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A}$ to a minimal model I' of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{CN}(\psi)$ by facts of the form $E_\chi(\mathbf{t})$, for χ a subformula of $\mathcal{C}$. By Lemma 4.6, $I' \models E_\mathcal{C}$ if and only if $I' \models \mathcal{C}$, which implies our claim. □

Theorem 4.8 is significant for two reasons: On the one hand, it provides the foundation for constraint checking the case when $\mathcal{S}$ is existential-free (see Section 6.2). On the other hand, it provides us with a slightly more procedural intuition about the nature of constraints. Rules of the form $A \rightarrow B$ from $\mathsf{LP}(\mathcal{S})$ do not contain negated atoms, so they can be seen as procedural rules of the form "from A conclude B." For each minimal set of facts derived from $\mathcal{A} \cup \mathcal{S}$, $\mathsf{CN}(\psi)$ simply checks whether $\mathcal{C}$ holds in it.

In subsequent sections, we shall occasionally abuse our notation and write $\mathsf{LP}(\mathcal{S})$ and $\mathsf{CN}(\mathcal{C})$ instead of $\mathsf{LP}(\pi(\mathcal{S}))$ and $\mathsf{CN}(\pi(\mathcal{C}))$.

4.3 Query Answering Under Constraints

We now present an important result about answering unions of positive conjunctive queries in extended DL knowledge bases. Namely, if the constraints are satisfied, then we need not consider them in query answering. This shows that our semantics for constraints is sensible: 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 extended DL knowledge bases, 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 4.9. *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.*[2] *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* $\pi(\mathcal{K}) \models \gamma(\mathbf{x})[\mathbf{c}/\mathbf{x}]$.

We next prove the following useful lemma:

Lemma 4.10. *Let* φ *be a first-order formula. If* $\mathsf{sk}(\varphi)$ *has a Herbrand model* I'*, then* $\mathsf{sk}(\varphi)$ *has a minimal Herbrand model* I *such that* $I \subseteq I'$.

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

Proof. The following property (*) is well-known: if a set of formulae has a Herbrand model, then it has a minimal Herbrand model as well. Such a model can be constructed, for example, using the model-construction method used to show the completeness of resolution [7]. Let I' be a Herbrand model of $\mathsf{sk}(\varphi)$, and let $S = \{\mathsf{sk}(\varphi)\} \cup \{\neg A \mid A \notin I'\}$. Clearly, S is satisfied in I'; furthermore, for each Herbrand model I'' of S, we have $I'' \subseteq I'$. Now by (*), a minimal Herbrand model I of S exists. Clearly, $I \subseteq I'$, and it is a minimal Herbrand model of $\mathsf{sk}(\varphi)$. □

We now prove the main result of this section:

Theorem 4.11. *Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ 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 $\mathcal{A} \cup \mathcal{S} \cup \mathcal{C} \models \gamma(\mathbf{c})$ if and only if $\mathcal{A} \cup \mathcal{S} \models \gamma(\mathbf{c})$.*

Proof. We show the contrapositive: if $\mathcal{K}$ satisfies $\mathcal{C}$, then $\mathcal{S} \cup \mathcal{A} \cup \mathcal{C} \not\models \gamma(\mathbf{c})$ if and only if $\mathcal{S} \cup \mathcal{A} \not\models \gamma(\mathbf{c})$. The ($\Rightarrow$) direction holds trivially by monotonicity, so we consider the ($\Leftarrow$) direction.

If $\mathcal{S} \cup \mathcal{A} \not\models \gamma(\mathbf{c})$, then $\mathsf{sk}(\mathcal{S} \cup \mathcal{A} \cup \{\neg\gamma(\mathbf{c})\})$ is satisfiable in a Herbrand model I'. The formula $\neg\gamma(\mathbf{c})$ is equivalent to $\bigwedge_{i=1}^{n} \forall \mathbf{y_i} : \neg Q_i(\mathbf{c}, \mathbf{y_i})$. It does not contain existential quantifiers, so it is not skolemized. Thus, we have $\mathsf{sk}(\mathcal{S} \cup \mathcal{A} \cup \{\neg\gamma(\mathbf{c})\}) = \mathsf{sk}(\mathcal{S} \cup \mathcal{A}) \cup \{\neg\gamma(\mathbf{c})\}$. By Lemma 4.10, a minimal Herbrand interpretation $I \subseteq I'$ exists such that $I \models \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$. Now $I' \not\models \gamma(\mathbf{c})$, so $I' \not\models \exists \mathbf{y_i} : Q_i(\mathbf{c}, \mathbf{y_i})$ for each $1 \leq i \leq n$. Hence, for each tuple $\mathbf{t}$ of the elements of the Herbrand universe, $I' \not\models Q_i(\mathbf{c}, \mathbf{y_i})[\mathbf{t}/\mathbf{y_i}]$. But then, since $I \subseteq I'$ and all the atoms from $Q_i(\mathbf{c}, \mathbf{y_i})$ are positive, we have $I \not\models Q_i(\mathbf{c}, \mathbf{y_i})[\mathbf{t}/\mathbf{y_i}]$ for each $\mathbf{t}$ as well, so $I \not\models \exists \mathbf{y_i} : Q_i(\mathbf{c}, \mathbf{y_i})$. Thus, we conclude $I \not\models \gamma(\mathbf{c})$. Since $\mathcal{K}$ satisfies $\mathcal{C}$, the latter is satisfied in each minimal Herbrand model of φ, so $I \models \mathcal{C}$. Hence, we conclude that $I \models \mathcal{S} \cup \mathcal{A} \cup \mathcal{C}$ and $I \not\models \gamma(\mathbf{c})$, which implies our claim. □

The proof of Theorem 4.11 shows why the conjunctive queries in $\gamma(\mathbf{x})$ must contain only positive atoms. Let $\gamma = \neg A(a)$ and consider a model I' such that $I' \models \mathsf{sk}(\mathcal{A} \cup \mathcal{S})$ and $I' \not\models \gamma$. For a minimal model I of $\mathsf{sk}(\mathcal{A} \cup \mathcal{S})$, it is not necessarily true that $I \not\models \gamma$: if $A(a) \in I' \setminus I$, then $I \models \neg A(a)$. Intuitively, constraint checking ensures that all facts derivable in a minimal model through constraints are derivable without constraints as well; however, this does not necessarily hold for negative formulae.

The proof of Theorem 4.11 also reveals why entailment of universally quantified formulae is not preserved. Intuitively, we cannot examine only the Herbrand models of $\mathsf{sk}(\mathcal{A} \cup \mathcal{S})$ for such formulae because they can be "too small." For example, let $\mathcal{A} = \{A(a)\}$ and $\varphi = \forall x : A(x)$. Clearly, $\mathcal{A} \not\models \varphi$, but the only Herbrand model of $\mathcal{A}$ is $I = \{A(a)\}$ and $I \models \varphi$. To use Herbrand models, we must reduce entailment to satisfiability. In doing

so, the universal quantifier in φ would become an existential quantifier, so skolemizing it would extend the Herbrand universe.

Theorem 4.11 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 for $\mathcal{K}$, 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, without treating $\mathcal{C}$ and $\mathcal{S}$ differently. 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 4 is very general. To provide practical guidance for modelers, we now discuss some typical usage patterns.

5.1 Participation Constraints

Participation constraints 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 $\bowtie \in \{\leq, \geq, =\}$:

$$C \sqsubseteq \bowtie n\, R.D \tag{46}$$

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 example of a participation constraint is (5), which states that each person must have an explicitly specified social security number. Another example is the following statement restricting each person to have at most one spouse:

$$Person \sqsubseteq \leq 1\, marriedTo.Person \tag{47}$$

To understand the difference in treating (47) a standard axiom or a constraint, consider an ABox $\mathcal{A}$ containing the following facts:

$$Person(Peter) \tag{48}$$

$$marriedTo(Peter, Ann) \tag{49}$$

$$marriedTo(Peter, Mary) \tag{50}$$

If (47) were a part of the standard TBox $\mathcal{S}$, then $\mathcal{A} \cup \mathcal{S}$ would be satisfiable; furthermore, due to (47), we would derive $Ann \approx Mary$. If we put (47) into

the constraint TBox $\mathcal{C}$, then the only minimal model of $\mathcal{A}$ contains exactly the facts (48)–(50). 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 (47) 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 a typical form 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

$$\exists R.\top \sqsubseteq C \tag{51}$$

whereas for range constraints it is

$$\top \sqsubseteq \forall R.C. \tag{52}$$

A typical example of a domain constraint is (13), 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*:

$$\top \sqsubseteq \forall \mathit{marriedTo}.\mathit{Person} \tag{53}$$

To understand the difference in treating (53) a standard axiom or a constraint, consider an ABox $\mathcal{A}$ containing only the fact (49). If (53) were a part of the standard TBox $\mathcal{S}$, then $\mathcal{A} \cup \mathcal{S}$ would be satisfiable; furthermore, due to (53), we would derive $\mathit{Person}(\mathit{Ann})$. If we put (53) into the constraint TBox $\mathcal{C}$, then the only minimal model of $\mathcal{A}$ contains only the fact (49). Thus, *Ann* is not explicitly typed to be a *Person*, so the constraint (53) 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 (5)–(7) 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 $\mathcal{S}$:

$$PersonTR \sqsubseteq Person \tag{54}$$

We require two things to 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:

$$PersonTR \sqsubseteq O \tag{55}$$

$$PersonTR \sqsubseteq \exists hasSSN.(O \sqcap SSN) \tag{56}$$

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

$$O(a) \tag{57}$$

Now in any minimal model of $\mathcal{S} \cup \mathcal{A}$, the assertions of the form (57) ensure that O is interpreted exactly as the set of all known objects. Hence, (55) ensures that each *PersonTR* is known, and (56) ensures that the social security number for each person is known as well.

One might criticize this solution on the grounds that it is not completely model-theoretic: it requires asserting (57) for each known individual, which is a form of "procedural preprocessing." We agree that the presented solution is not completely clean in that sense; however, we believe that it is simple to understand and implement, which outweighs its drawbacks.

For TBox reasoning, we do not assert any statement of the form (57). Namely, during TBox reasoning, we are considering all possible ABoxes, so O can be interpreted as an arbitrary subset of the interpretation domain.

Finally, instead of the axioms (57), one might intuitively expect O to be defined as a nominal containing all the individuals from the ABox:

$$O \equiv \{a_1, \ldots, a_n\} \tag{58}$$

This, however, requires nominals to be available in the DL language, which is known to make reasoning more difficult. Furthermore, if O does not occur in the standard TBox axioms, but only in the constraint axioms, then axioms of the form (57) are sufficient: the minimal model semantics of constraints ensures that O contains exactly the specified individuals.

6 Checking Constraints

We now turn our attention to the problem of checking whether an extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ satisfies the constraint TBox $\mathcal{C}$. The difficulty in checking satisfiability of constraints is determined by the structure of $\mathcal{S}$. Namely, evaluating a formula in a Herbrand model is easy regardless of the formula structure; the difficult task is computing the minimal models of $\mathsf{sk}(\mathcal{A} \cup \mathcal{S})$. In the rest of this section, we consider different possibilities for doing so depending on the form of $\mathcal{S}$.

If $\mathcal{S}$ contains neither existential quantifiers under positive nor universal quantifiers under negative polarity, we can use Theorem 4.8: the program $\mathcal{A} \cup \mathsf{LP}(\mathcal{S}) \cup \mathsf{CN}(\mathcal{C})$ does not contain function symbols, so we can use any (disjunctive) datalog engine. A minor difficulty is caused by the fact that $\mathsf{LP}(\mathcal{S})$ can be exponential in size. Therefore, in Section 6.1, we show how to perform the translation without a blowup, and, in Section 6.2, we apply this result to *existential-free* knowledge bases. We consider the more general case in Section 6.3, where we present a procedure for the case where $\mathcal{S}$ is an $\mathcal{ALCHI}$ knowledge base.

6.1 The Structural Transformation

According to Definition 4.7, the logic program $\mathsf{LP}(\mathcal{S})$ is computed by skolemizing $\pi(\mathcal{S})$ and translating it into conjunctive normal form. It is well known that applying the distributivity laws for conjunction and disjunction can incur an exponential blowup, so Definition 4.7 yields an inefficient algorithm. This problem can be remedied by applying the *structural transformation* [34, 33]. We next present the definition of the structural transformation here for the sake of completeness. In the definition, we use the well-known notions of positions and polarity. With $\varphi|_p$ we denote the subformula of a formula φ at position p, and with $\varphi[\psi]_p$ the formula obtained by replacing the subformula of φ at position p with a formula ψ.

Definition 6.1. *An application of* structural transformation *to a position p in a formula φ is the formula $\mathsf{st}(\varphi, p)$ defined as follows, where $\mathbf{x}$ is the set of free variables of $\varphi|_p$ and Q is a predicate not occurring in φ:*

$$\mathsf{st}(\varphi, p) = \begin{cases} \varphi[Q(\mathbf{x})]_p \wedge (Q(\mathbf{x}) \rightarrow \varphi|_p) & \text{if the polarity of } \varphi|_p \text{ in } \varphi \text{ is positive} \\ \varphi[Q(\mathbf{x})]_p \wedge (Q(\mathbf{x}) \leftarrow \varphi|_p) & \text{if the polarity of } \varphi|_p \text{ in } \varphi \text{ is negative} \\ \varphi[Q(\mathbf{x})]_p \wedge (Q(\mathbf{x}) \leftrightarrow \varphi|_p) & \text{otherwise} \end{cases}$$

Structural transformation is typically applied to a formula recursively until it can be transformed into conjunctive normal form without an exponential blowup. For example, in order to avoid distributing the conjunction over the disjunction, the formula

$$\forall x : [A(x) \rightarrow \exists y : R(x, y) \wedge (B(y) \vee C(y))]$$

is transformed into a formula

$$\forall x : [A(x) \rightarrow \exists y : R(x, y) \wedge Q(y)] \wedge \forall y : [Q(y) \rightarrow B(y) \vee C(y)]$$

that can be converted into conjunctive normal form in a straightforward way. It is well-known that structural transformation preserves satisfiability and that it can be applied to a formula in polynomial time.

Since it extends the signature of the formula, structural transformation clearly does not preserve the models of a formula. For an interpretation I and a set of predicates Υ, let $I/_\Upsilon$ be the restriction of I to the predicates in S, defined as follows:

$$I/_\Upsilon = \{A(t_1, \ldots, t_n) \in I \mid A \in \Upsilon\}$$

For a formula φ, with $I/_\varphi$ we denote $I/_\Upsilon$ where Υ is the set of all predicates occurring in φ. Let φ be a formula and ψ a formula obtained from φ through structural transformation. Ideally, we would like each minimal model I of φ to have a counterpart minimal model I' of ψ such that $I = I'/_\varphi$; conversely, for each minimal model I' of ψ, we would like $I'/_\varphi$ to be a minimal model of φ. Unfortunately, this does not hold. Let φ_1 be the following formula:

$$\varphi_1 = A \wedge C \wedge [A \rightarrow B \vee (C \wedge \neg D)]$$

The only minimal model of φ_1 is $I = \{A, C\}$. Furthermore, applying structural transformation to φ_1 produces the following formula:

$$\psi_1 = A \wedge C \wedge (A \rightarrow B \vee Q) \wedge (Q \rightarrow C \wedge \neg D)$$

The minimal models of ψ_1 are $I'_1 = \{A, Q, C\}$ and $I'_2 = \{A, B, C\}$. Now $I'_1/_{\varphi_1} = I$, which is as expected; however, I'_2 does not correspond to a minimal model of φ.

To describe the correspondence between the models of formulae before and after the structural transformation, we use the following definition. For a set of predicates Υ, a Herbrand interpretation I is an Υ-*minimal model* of a formula ψ if $I \models \psi$ and $I' \not\models \psi$ for each interpretation I' such that $I'/_\Upsilon \subset I/_\Upsilon$. A φ-*minimal model* of ψ is the Υ-minimal model of ψ where Υ is the set of all predicates of φ.

Theorem 6.2. *For any first-order formula φ and $\psi = \mathsf{st}(\varphi, p)$ for some position p, the following two claims hold:*

- *For each minimal Herbrand model I of φ, a minimal model I' of ψ exists such that $I = I'/_\varphi$.*
- *Conversely, $I'/_\varphi$ is a minimal Herbrand model of φ for each φ-minimal Herbrand model I' of ψ.*

Proof. Let φ, ψ, and χ be as stated in the theorem. The following properties are well-known [33]: (*) for each model I' of ψ, we have $I'/_\varphi \models \varphi$; and (**) for each model I of φ, a model I'' of ψ exists such that $I''/_\varphi = I$.

(Claim 1.) Let I be a minimal Herbrand model of φ, and let I'' be a Herbrand model whose existence is implied by (**). Clearly, ψ must have a minimal Herbrand model I' such that $I \subseteq I' \subseteq I''$ and $I'/_\varphi = I$.

(Claim 2.) Let I' be a φ-minimal Herbrand model of ψ. By (*), $I'/_\varphi \models \varphi$. Let us assume that $I'/_\varphi$ is not a minimal model of φ—that is, that an interpretation I exists such that $I \subsetneq I'/_\varphi$ and $I \models \varphi$. But then, by the first claim, a minimal model I'' of ψ exists such that $I''/_\varphi = I$ and $I''/_\varphi \subsetneq I'/_\varphi$. Hence, I' is not a φ-minimal model of ψ. □

Note that I'' from the proof of Claim 1 does not need to be a minimal model of ψ. For example, let φ_2 be the following formula:

$$\varphi_2 = A \wedge B \wedge C \wedge [A \rightarrow B \vee (C \wedge \neg D)]$$

By applying the structural transformation to $C \wedge \neg D$, we get the following formula ψ_2:

$$\psi_2 = A \wedge B \wedge C \wedge (A \rightarrow B \vee Q) \wedge (Q \rightarrow C \wedge \neg D)$$

The only minimal model of φ_2 is $I = \{A, B, C\}$. Furthermore, $I'' = I \cup \{Q\}$ is a model of ψ_2, but it is not a minimal one: it is not necessary to make Q true since A, B, and C are already true. Theorem 6.2 is based on the observation that I'' is a model, so ψ_2 must have a minimal model that is somewhere between I and I''.

Certain description logics (e.g., Horn-$\mathcal{SHIQ}$ [25], DL-lite [12], or $\mathcal{EL}++$ [2]) can be translated into a first-order logic formula φ that can be translated into a conjunction of disjunctions with at most one positive literal. In such cases, the structural transformation does not affect the minimality of models:

Proposition 6.3. *Let φ be a first-order formula that can be converted using structural transformation into a formula $\psi = \bigwedge C_i$ where $C_i = \bigvee L_{ij}$ and each C_i contains at most one positive literal. If I is a minimal model of ψ, then $I/_\varphi$ is a minimal model of φ.*

Proof. This claim follows immediately from Theorem 6.2 and the fact that ψ is a conjunction of Horn clauses which is known to have at most one minimal model. □

6.2 Checking Constraints for Existential-Free KBs

We now consider checking satisfaction of constraints for so-called existential-free knowledge bases:

Definition 6.4. *A knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ is* existential-free *if $\pi(\mathcal{S})$ contains neither existential quantifiers under positive nor universal quantifiers under negative polarity.*

For example, $\mathcal{S}$ is allowed to contain an axiom of the form $\exists R.C \sqsubseteq D$ (the existential quantifier occurs on the left-hand side of the inclusion and is effectively equivalent to a universal quantifier), but not an axiom of the form $C \sqsubseteq \exists R.D$ (the existential quantifier now indeed implies existence of individuals in a model).

Function symbols in $\mathsf{LP}(\mathcal{S})$ are introduced only by skolemizing existential quantifiers under positive or universal quantifiers under negative polarity. Therefore, existential-free knowledge bases exhibit the following property:

Proposition 6.5. *For a knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$, the program $\mathsf{LP}(\mathcal{S})$ does not contain function symbols.*

Thus, for existential-free KBs, we can check constraint satisfaction using any reasoning engine for function-free disjunctive logic programs, such as DLV [16] or Smodels [39]. To check constraint satisfaction, we just compute $\mathcal{A} \cup \mathsf{LP}(\mathcal{S}) \cup \mathsf{CN}(\mathcal{C})$ and apply Theorem 4.8. Computing $\mathsf{LP}(\mathcal{S})$ can, in general, increase the size of the formula exponentially. To avoid this, we can apply the structural transformation to $\pi(\mathcal{S})$ before computing $\mathsf{LP}(\mathcal{S})$. If $\mathcal{S}$ can be translated into Horn logic, then the structural transformation does not change the minimal models of $\mathsf{LP}(\mathcal{S})$ by Proposition 6.3, so Theorem 4.8 holds without any change. The data complexity of query answering in a stratified datalog program is polynomial [14], which immediately implies that the satisfaction of constraints can be checked for Horn existential-free knowledge bases with polynomial data complexity.

In the general case when $\mathcal{S}$ cannot be translated into Horn logic, structural transformation can introduce new minimal models. Then, we can apply Theorem 6.2: we can use DLV or Smodels to compute all minimal models of $\mathcal{A} \cup \mathsf{LP}(\mathcal{S}) \cup \mathsf{CN}(\mathcal{C})$, filter out all $\pi(\mathcal{S})$-minimal models, and then evaluate $\mathsf{CN}(\mathcal{C})$ in them.

Filtering out the $\pi(\mathcal{S})$-minimal models can also be built directly into the reasoning algorithm of the logic programming engines. Let φ be a propositional formula (all disjunctive datalog engines known to us ground the program, and thus reduce the problem to a propositional one) and I a propositional interpretation such that $I \models \varphi$. It is well-known [32] that I is a minimal model of φ if and only if $\chi(\varphi, I)$, defined as follows, is unsatisfiable:

$$\begin{aligned} \chi(\varphi, I) &= \varphi \wedge \mathsf{neg}(I) \wedge \mathsf{pos}(I) \\ \mathsf{neg}(I) &= \bigwedge_{A \notin I} \neg A \\ \mathsf{pos}(I) &= \bigvee_{A \in I} \neg A \end{aligned}$$

Namely, $\mathsf{neg}(I)$ requires all negative atoms from I to be also negative in a model of $\chi(\varphi, I)$, and $\mathsf{pos}(I)$ requires at least one of the positive atoms from I to be negative in a model of $\chi(\varphi, I)$. We modify this check as follows:

Theorem 6.6. *Let Υ be a set of propositions, φ a propositional formula, and I an interpretation such that $I \models \varphi$. Then, I is an Υ-minimal model of φ if and only if $\chi(\varphi, I, \Upsilon)$, defined as follows, is unsatisfiable:*

$$\begin{aligned} \chi(\varphi, I, \Upsilon) &= \varphi \wedge \mathsf{neg}(I, \Upsilon) \wedge \mathsf{pos}(I, \Upsilon) \\ \mathsf{neg}(I, \Upsilon) &= \bigwedge_{A \in \Upsilon \setminus I} \neg A \\ \mathsf{pos}(I, \Upsilon) &= \bigvee_{A \in \Upsilon \cap I} \neg A \end{aligned}$$

Proof. ($\Rightarrow$) Assume that $\chi(\varphi, I, \Upsilon)$ is satisfiable in a model I'. Due to $\mathsf{neg}(I, \Upsilon)$, if $I \not\models A$, then $I' \not\models A$ as well; furthermore, due to $\mathsf{pos}(I, \Upsilon)$, there is at least one atom $I \models B$ such that $I' \not\models B$. Hence, I' is a model of φ and $I'/_\Upsilon \subset I/_\Upsilon$, so I is not an Υ-minimal model.

($\Leftarrow$) Assume that I is not a Υ-minimal model of φ. Then, a model I' of φ exists such that $I'/_\Upsilon \subset I/_\Upsilon$. Clearly, I' satisfies both $\mathsf{neg}(I, \Upsilon)$ and $\mathsf{pos}(I, \Upsilon)$, so $\chi(\varphi, I, \Upsilon)$ is satisfiable. □

Theorem 6.6 immediately implies Π_2^p as the upper bound on the data complexity of checking constraint satisfaction. Namely, the constraints are not satisfied if a minimal model I of $\mathcal{A} \cup \mathcal{S}$ exists that does not satisfy $\mathcal{C}$. Now I can be guessed in polynomial time and, due to Theorem 6.6, minimality can be checked with an oracle in NP.

We finish this section with a note about equality. Namely, most existing implementations of disjunctive logic programs support equality as a built-in predicate that is interpreted as identity and is allowed only to occur in the rule bodies. The program $\mathsf{LP}(\mathcal{S})$, however, can contain equality in the rule heads as well. This type of equality is traditionally not supported in logic programming; however, it can be simulated by introducing a new predicate and explicitly axiomatizing the equality properties for it [17]. Note that the logic program $\mathsf{CN}(\mathcal{C})$ can also contain equality, but only in the rule bodies. Hence, $\mathsf{CN}(\mathcal{C})$ cannot constrain two constants to be equal; it can only check whether the two constraints have been derived to be equal. If $\mathsf{CN}(\mathcal{C})$ contains equality but $\mathsf{LP}(\mathcal{S})$ does not, then we can simply interpret equality in $\mathsf{CN}(\mathcal{C})$ as identity and use the built-in implementation of logic programming engines.

6.3 Checking Constraints in the Presence of Existentials

We now turn our attention to the problem of checking constraints in the case when the standard TBox $\mathcal{S}$ contains existential quantifiers under positive or universal quantifiers under negative polarity. The Herbrand models

of $\mathsf{sk}(\mathcal{A} \cup \mathcal{S})$ are then infinite, so we cannot represent them directly. The standard tableau reasoning algorithms deal with this problem by generating a finite abstraction of a model, so one might attempt to extend these algorithms to support minimal model reasoning. Unfortunately, this proves to be quite a difficult task: since the models are not fully represented, it is difficult to compare them for minimality.

Constraint checking is, however, easier for logics having the tree-model property. In this section, we present a decision procedure for the case when the standard TBox $\mathcal{S}$ is expressed in the DL $\mathcal{ALCHI}$. This logic contains many characteristic DL constructs: the Boolean connectives $\neg C$, $C \sqcup D$, and $C \sqcap D$, existential quantification $\exists R.C$, universal quantification $\forall R.C$, inverse roles R^-, and role hierarchies $R \sqsubseteq S$. The semantics of an $\mathcal{ALCHI}$ knowledge base $\mathcal{K}$ is given by translating it into a first-order formula $\pi(\mathcal{K})$ as specified in Table 2. We conjecture that the procedure presented here can be extended to handle qualified number restrictions as well; however, the presence of qualified number restrictions introduces technical problems which obscure the nature of our result. Furthermore, the presented algorithm is not intended to be used in practice; rather, it should be understood as evidence that checking constraints is, in principle, possible in the presence of existentials for nontrivial description logics. We also conjecture that constraint checking in the case where $\mathcal{S}$ contains constructors such as nominals, transitive roles, or generalized role inclusion axioms [26] is decidable; proving this is, however, nontrivial, and we leave it for future work. We do not restrict the structure of the constraint TBox $\mathcal{C}$ in any way. Thus, the combined TBox $\mathcal{S} \cup \mathcal{C}$ can be more expressive than $\mathcal{ALCHI}$.

We embed the constraint satisfaction checking problem into the second-order monadic logic on infinite k-ary trees *SkS* [35, 40]. This logic provides for second-order quantification over unary predicates, which allows us to easily express the minimality criterion. Furthermore, we should be able to build an automata-based decision procedure and then, along the lines of [5], derive a practical tableau procedure. We briefly overview this logic next. In general, we follow the presentation from [40]; however, instead of binary predicates, we represent the successor relationship by function symbols.

Let k be a positive integer. *SkS* terms are built from first-order variables (written as x, y, z, etc.), a constant symbol ε, and k unary function symbols $f_i(t)$, as usual. For *SkS* terms t and s, an *SkS* atom is of the form $t = s$ or $X(t)$, where X is a second-order variable (we write all such variables in capital letters). *SkS* formulae are obtained from atoms in the usual way using propositional connectives $\wedge$, $\vee$, and $\neg$, first-order quantification $\exists x$ and $\forall x$, and second-order quantification $\exists X$ and $\forall X$. For the semantics of *SkS*, please refer to [40]. Intuitively, first-order quantification ranges over domain elements, whereas second-order quantification ranges over domain subsets. The symbol $=$ denotes true equality in *SkS*; it is different from the symbol $\approx$, which denotes a congruence relation on Herbrand models.

Table 2: Semantics of $\mathcal{ALCHI}$ by Mapping to FOL

Mapping Roles to FOL	
$\pi_{xy}(R) = R(x,y)$	$\pi_{yx}(R) = R(y,x)$
$\pi_{xy}(R^-) = R(y,x)$	$\pi_{yx}(R^-) = R(x,y)$
Mapping Concepts to FOL	
$\pi_x(A) = A(x)$	$\pi_y(A) = A(y)$
$\pi_x(\neg C) = \neg\pi_x(C)$	$\pi_y(\neg C) = \neg\pi_y(C)$
$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$	$\pi_y(C \sqcap D) = \pi_y(C) \wedge \pi_y(D)$
$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$	$\pi_y(C \sqcup D) = \pi_y(C) \vee \pi_y(D)$
$\pi_x(\exists R.C) = \exists y : \pi_{xy}(R) \wedge \pi_y(C)$	$\pi_y(\exists R.C) = \exists x : \pi_{yx}(R) \wedge \pi_x(C)$
$\pi_x(\forall R.C) = \forall y : \pi_{xy}(R) \rightarrow \pi_y(C)$	$\pi_y(\forall R.C) = \forall x : \pi_{yx}(R) \rightarrow \pi_x(C)$
Mapping Axioms to FOL	
$\pi(C \sqsubseteq D) = \forall x : \pi_x(C) \rightarrow \pi_x(D)$	
$\pi(R \sqsubseteq S) = \forall x, y : \pi_{xy}(R) \rightarrow \pi_{xy}(S)$	
$\pi(A(a)) = A(a)$	
$\pi(R(a,b)) = R(a,b)$	

Checking satisfiability of a closed *SkS* formula φ can be performed in time

$$2^{\overset{\overset{O(n)}{\cdots}}{2}}\Big\}q+1$$

where n is the length of φ and q is the number of quantifier changes in the prenex normal form of φ [40, page 217]. We use $\varphi_1 \rightarrow \varphi_2$ as a syntactic shortcut for $\neg\varphi_1 \vee \varphi_2$; $P \subseteq R$ as a syntactic shortcut for $\forall x : P(x) \rightarrow R(x)$; and $P \subset R$ for $P \subseteq R \wedge \neg(R \subseteq P)$.

Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base. Next, we compute an *SkS* formula that is satisfiable if and only if $\mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S})) \models_{\mathsf{MM}} \pi(\mathcal{C})$. To compute $\psi = \mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S}))$, we first bring $\pi(\mathcal{A} \cup \mathcal{S})$ into negation-normal form. Without loss of generality, we assume that the roles of the form $(R^-)^-$ are replaced with R. By definition, ψ then has the following form, where λ is as defined in Table 3:

$$\begin{aligned}
\psi &= \mathcal{A} \wedge \psi_1 \wedge \psi_2 \\
\psi_1 &= \bigwedge_{R \sqsubseteq S \in \mathcal{S}} \forall x, y : [\pi_{xy}(R) \rightarrow \pi_{xy}(S)] \\
\psi_2 &= \bigwedge_{C \sqsubseteq D \in \mathcal{S}} \forall x : \lambda(\mathsf{NNF}(\neg C \sqcup D), x)
\end{aligned}$$

Table 3: Skolemization of Concepts in Negation-Normal Form

C	$\lambda(C, \cdot)$
A	$A(\cdot)$
$\neg A$	$\neg\lambda(A, \cdot)$
$C_1 \sqcap C_2$	$\lambda(C_1, \cdot) \wedge \lambda(C_2, \cdot)$
$C_1 \sqcup C_2$	$\lambda(C_1, \cdot) \vee \lambda(C_2, \cdot)$
$\exists R.C$	$R(\cdot, f(\cdot)) \wedge \lambda(C, f(\cdot))$
$\exists R^-.C$	$R(f(\cdot), \cdot) \wedge \lambda(C, f(\cdot))$
$\forall R.C$	$\forall y : [R(\cdot, y) \rightarrow \lambda(C, y)]$
$\forall R^-.C$	$\forall y : [R(y, \cdot) \rightarrow \lambda(C, y)]$

Note: The symbol $\cdot$ is a placeholder for actual terms supplied as the second argument to λ. The function symbol f and the variable y are new in each invocation of λ.

The formula ψ is not an *SkS* formula because it contains binary atoms. Next, we show that we can encode those atoms using unary atoms. For this, we show that all minimal models of ψ are forest-like, as defined next:

Definition 6.7. *A Herbrand interpretation I is* forest-like *if it contains only unary and binary atoms, all function symbols are at most unary, and all binary atoms are of the form $R(a, t)$, $R(t, f(t))$, or $R(f(t), t)$, where a is a constant and t is a term.*

One might expect forest-like models to contain the facts of the form $R(a, b)$ instead of the facts of the form $R(a, t)$; we discuss the reason for this after Definition 6.10.

Lemma 6.8. *Each minimal model of $\mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S}))$ is forest-like.*

Proof. If I is a model of $\psi = \mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S}))$ that is not forest-like, it contains a binary atom of the form $R(s, t)$ that is not of the form specified in Definition 6.7. Let I' be an interpretation obtained from I by removing all atoms of the form $S(s, t)$ such that $S \sqsubseteq^* R$, where $\sqsubseteq^*$ is the reflexive-transitive closure of the role inclusion relation $\sqsubseteq$. For the subformula ψ_1 of ψ, it is clear that $I \models \psi_1$ if and only if $I' \models \psi_1$ because, whenever we remove some $R(s, t)$ from I, we remove also all $S(s, t)$ such that $S \sqsubseteq^* R$. For the subformula ψ_2 of ψ, we show that $I \models \psi_2$ if and only if $I' \models \psi_2$ by a straightforward induction on the formula structure. Namely, only positive binary atoms could have a different truth value in I and I'; however, all such atoms in ψ_2 are of the form $R(t, f(t))$ or $R(f(t), t)$, and they are included in I' whenever they are included in I. □

Our goal is to represent each forest-like model using a model of the following form:

Table 4: Transforming Interpretations

Forest-like		Tree-like
$R(t, f(t))$	$\leftrightarrow$	$R_f(t)$
$R(f(t), t)$	$\leftrightarrow$	$R_f^-(t)$
$R(a, t)$	$\leftrightarrow$	$R_a(t)$ for t not of the form $f(a)$

Definition 6.9. *A Herbrand interpretation I is* monadic *if it contains only unary predicates and all function symbols are at most unary.*

To encode the forest-like models using the monadic models, we introduce the unary predicates R_f, R_f^-, and R_a for each binary predicate R, each function symbol f, and each constant a. We use these predicates to encode binary atoms in a forest-like model, as shown in the following definition:

Definition 6.10. *For I a forest-like Herbrand interpretation, the* monadic encoding $\widetilde{I}$ *is obtained by replacing each atom from the left-hand side of Table 4 with the corresponding atom on the right-hand side. For I a monadic Herbrand interpretation, the* forest-like encoding $\overline{I}$ *is obtained by replacing each atom from the right-hand side of Table 4 with the corresponding atom on the left-hand side.*

We clarify an important point about the previous definition. For consistency, we might be tempted to use the predicates R_f and R_f^- to always encode the starting points of binary relations; then, we would encode $R(f(t), t)$ as $R_f^-(f(t))$. But then, we would lose the one-to-one correspondence between forest-like and monadic interpretations; for example, a monadic interpretation containing an atom $R_f^-(a)$ does not correspond to any forest-like interpretation because there is no predecessor for a. Definition 6.10 places the predicates R_f or R_f^- on the node closer to the tree root, so, when "decoding" a monadic interpretation, we can always find the appropriate successor.

Similarly, we might restrict the forest-like interpretations only to atoms of the form $R(a, b)$ instead of $R(a, t)$. But then, since "decoding" an atom $R_a(f(a))$ from a monadic interpretation produces an atom $R(a, f(a))$, we would lose the one-to-one correspondence between forest-like and monadic interpretations. Therefore, Definition 6.10 places the predicate R_a on the varying endpoints of binary atoms of the form $R(a, t)$.

We now show how to encode binary literals using unary literals in an arbitrary formula. We remind the reader that $=$ is true equality, different from the congruence relation on Herbrand models $\approx$.

Definition 6.11. *For a binary predicate R and a set of symbols Σ, the formula $\nu[R,\Sigma](x,y)$ is defined as follows:*

$$\begin{aligned}
\nu[R,\Sigma](x,y) &= \nu_1[R,\Sigma](x,y) \vee \nu_2[R,\Sigma](x,y) \vee \nu_3[R,\Sigma](x,y) \\
\nu_1[R,\Sigma](x,y) &= \bigvee_{R,a\in\Sigma} [x = a \wedge R_a(y)] \\
\nu_2[R,\Sigma](x,y) &= \bigvee_{R,f\in\Sigma} [y = f(x) \wedge R_f(x)] \\
\nu_3[R,\Sigma](x,y) &= \bigvee_{R,f\in\Sigma} [x = f(y) \wedge R_f^-(y)]
\end{aligned}$$

For a formula φ, the formula $\nu[\varphi,\Sigma]$ is obtained from φ by replacing each atom $R(s,t)$ with $\nu[R,\Sigma](s,t)$.[3]

We next show that the encoding from Definition 6.11 preserves validity of a formula in a forest-like model:

Lemma 6.12. *Let $\widetilde{I}$ and $\overline{J}$ be defined as in Definition 6.10. Furthermore, let φ be a formula containing only unary and binary predicates, Σ a set of containing all constants and binary predicates from φ, and $\xi = \nu[\varphi,\Sigma]$. Then, the following two claims hold:*

- *For I a forest-like Herbrand interpretation, $I \models \varphi$ implies $\widetilde{I} \models \xi$.*
- *For J a monadic Herbrand interpretation, $J \models \xi$ implies $\overline{J} \models \varphi$.*

Proof. (Claim 1.) We prove a slightly more general claim. Let φ be a formula containing only unary and binary predicates with free variables $x_1,\ldots,x_n$, and let $\xi = \nu[\varphi,\Sigma]$. Then, for any assignment σ of the variables x_i to ground terms t_i, we have $I \models \varphi\sigma$ if and only if $\widetilde{I} \models \xi\sigma$. The proof is by induction on the structure of φ. The base case for unary atoms is trivial, since I and $\widetilde{I}$ coincide on unary atoms.

Let $\varphi = R(u,v)$; the formula ξ is of the form as in Definition 6.11. Since I is forest-like, $I \models R(u\sigma,v\sigma)$ if and only if $R(u\sigma,v\sigma)$ is of the form $R(a,t)$, $R(t,f(t))$, or $R(f(t),t)$. In the first case, $\widetilde{I} \models \nu_1[R](u\sigma,v\sigma)$; in the second case, $\widetilde{I} \models \nu_2[R](u\sigma,v\sigma)$; and in the third case, $\widetilde{I} \models \nu_3[R](u\sigma,v\sigma)$.

The induction step for Boolean connectives and quantifies is trivial and is omitted for the sake of brevity.

(Claim 2.) The proof of this claim is completely analogous to the proof of the first claim. □

Our final obstacle is caused by the fact that *SkS* provides for only one constant ε, whereas ψ can contain n different constants $a_1,\ldots,a_n$. Therefore, we must encode these constants using function symbols, as shown next.

[3]Note that φ does not contain inverse role atoms $R^-(s,t)$; such atoms are replaced with $R(t,s)$ as shown in Table 2.

Definition 6.13. *Let ψ be a skolemized first-order formula containing unary function symbols $f_1, \ldots, f_m$ and constants $a_1, \ldots, a_n$, and not containing the constant ε. Let $f_{m+1}, \ldots, f_k$ be new unary function symbols not occurring in ψ, for $k = n + m$. Furthermore, let φ be a first-order formula containing only constants from ψ. Then, $\mathsf{cs}_\varphi(\psi)$ is the formula obtained from φ by replacing each constant a_i with $f_{m+i}(\varepsilon)$.*

In the rest of this section, we use a_i as a syntactic shortcut for $f_{m+i}(\varepsilon)$. The following proposition follows trivially from the fact that ε and Let $f_{m+1}, \ldots, f_k$ do not occur in ψ:

Proposition 6.14. *Let ψ and φ be formulae as in Definition 6.13. Each minimal Herbrand model I of ψ corresponds to exactly one minimal Herbrand model I' of $\mathsf{cs}_\psi(\psi)$ and vice versa. Furthermore, for such I and I', we have $I \models \varphi$ if and only if $I' \models \mathsf{cs}_\psi(\varphi)$.*

We are now ready to define an algorithm for checking satisfaction of constraints in an extended DL knowledge base $\mathcal{K}$. We construct a formula $SkS_\mathcal{K}$ that is satisfiable if and only if the constraints are satisfied. Intuitively, the outer quantifiers $\forall P_1, \ldots, P_n$ in $SkS_\mathcal{K}$ define a valuation I of propositional symbols; then, the formula SkS_α evaluates $\mathcal{A} \cup \mathcal{S}$ in I; next, the formula SkS_{MM} ensures that I is a minimal model for $\mathcal{A} \cup \mathcal{S}$; and, finally, the formula SkS_β evaluates $\mathcal{C}$ in I.

Theorem 6.15. *Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base such that $\mathcal{S}$ is expressed in the DL $\mathcal{ALCHI}$. Furthermore, let $\psi = \mathsf{sk}(\pi(\mathcal{A} \cup \mathcal{S}))$, $\alpha = \mathsf{cs}_\psi(\psi)$, $\varphi = \pi(\mathcal{C})$, and $\beta = \mathsf{cs}_\psi(\varphi)$. Finally, let Σ be the set of all binary predicates and constants from ψ. With α' we denote the formula obtained from α by replacing each predicate P with a new predicate P'. Finally, let $SkS_\mathcal{K}$ be the following SkS formula, where P_i are all predicates of SkS_ψ, and P'_i are all predicates of $SkS_{\alpha'}$:*

$$
\begin{aligned}
SkS_\mathcal{K} &= \forall P_1, \ldots, P_n : [(SkS_\alpha \wedge SkS_{MM}) \rightarrow SkS_\beta] \\
SkS_\alpha &= \nu[\alpha, \Sigma] \\
SkS_\beta &= \nu[\beta, \Sigma] \\
SkS_\subset &= (P'_1 \subseteq P_1 \wedge \ldots \wedge P'_n \subseteq P_n) \wedge (P'_1 \subset P_1 \vee \ldots \vee P'_n \subset P_n) \\
SkS_{MM} &= \forall P'_1, \ldots, P'_n : SkS_\subset \rightarrow \neg SkS_{\alpha'} \\
SkS_{\alpha'} &= \nu[\alpha', \Sigma]
\end{aligned}
$$

Then, $\psi \models_{\mathsf{MM}} \varphi$ if and only $SkS_\mathcal{K}$ is valid.

Proof. ($\Rightarrow$) If $SkS_\mathcal{K}$ is not valid, a monadic interpretation I of the predicates P_i exists such that $I \models SkS_\alpha$ and $I \models SkS_{MM}$, but $I \not\models SkS_\beta$. By Lemma 6.12, $\overline{I} \models \alpha$ and $\overline{I} \not\models \beta$. We next show that $\overline{I}$ is a minimal model of α, which implies that $\alpha \not\models_{\mathsf{MM}} \beta$; by Proposition 6.14, we then have that $\psi \not\models_{\mathsf{MM}} \varphi$.

Assume that $\overline{I}$ is not a minimal model of α—that is, that an interpretation $J \subset \overline{I}$ exists such that $J \models \alpha$. By Lemma 6.8, J is forest-like. But then, $\widetilde{J} \subset I$, which implies that $\widetilde{J} \models SkS_{\subset}$; furthermore, by Lemma 6.12, $\widetilde{J} \models SkS_{\alpha'}$. These two claims now imply that $I \not\models SkS_{MM}$, which is a contradiction.

($\Leftarrow$) If $\psi \not\models_{\mathsf{MM}} \varphi$, by Proposition 6.14, we have that $\alpha \not\models_{\mathsf{MM}} \beta$. But then, by Lemma 6.8, a forest-like model I of α exists such that $I \not\models \beta$. By Lemma 6.12, $\widetilde{I} \models SkS_{\alpha}$ and $\widetilde{I} \not\models SkS_{\beta}$. To complete the proof that $SkS_{\mathcal{K}}$ is not valid, we just need to show that $\widetilde{I} \models SkS_{MM}$. Assume that the latter is not the case; then, a monadic interpretation J exists such that $J \subset \widetilde{I}$ and $J \models SkS_{\alpha'}$. But then, by Lemma 6.12, $\overline{J} \models \alpha'$ and $\overline{J} \subset I$, which implies that I is not a minimal model of α. □

Theorem 6.15 shows that checking constraint satisfaction is decidable for nontrivial description logics. Unfortunately, it gives us only a nonelementary upper complexity bound: the complexity is determined by the number of quantifier alternations, which is unlimited because SkS_{β} can be any first-order formula. We conjecture, however, that the complexity actually depends on the number of alterations of second-order quantifiers, and not of first-order quantifiers. In our future work, we shall try to see whether this holds, and derive tight complexity bounds.

7 Relationship to Autoepistemic DLs

The usefulness of constraint languages has been recognized early on in the knowledge representation community. In [36], Reiter noticed that constraints are epistemic in nature; furthermore, he presented an extension of first-order logics with an autoepistemic knowledge operator **K** that provides for introspection. Furthermore, in [28], Lifschitz presented the logic of Minimal Knowledge and Negation-as-Failure (MKNF) which, additionally, provides for a negation-as-failure operator **not**.

MKNF was used in [15] to obtain an expressive, but yet decidable nonmonotonic DL. One of the motivations for this work was to provide a language capable of expressing integrity constraints. For example, the constraint (5) can be expressed using the following axiom (the modal operator **A** corresponds to $\neg\,\mathbf{not}$ in MKNF):

$$\mathbf{K}\,Person \sqsubseteq \exists\,\mathbf{A}\,hasSSN.\,\mathbf{A}\,Person \tag{59}$$

MKNF was also used in [31] to integrate DLs with logic programming. Again, one of the motivations for this work was to allow for constraint modeling. For example, the constraint (5) can expressed using the following

logic programming rules:

$$\mathbf{K}\, OK(x) \leftarrow \mathbf{K}\, hasSSN(x, y), \mathbf{K}\, SSN(y) \tag{60}$$

$$\leftarrow \mathbf{K}\, Person(x), \mathbf{not}\, OK(x) \tag{61}$$

Although the motivation is the same, these approaches differ from the one presented in this paper in several important aspects. First, the rules (60)–(61) do not have any meaning during TBox reasoning; they can only be used to check whether an ABox is of a required shape. Furthermore, the axiom (59) might be applied during TBox reasoning, but it has a significantly different semantics: it can be applied only to the consequences of other modal axioms, and not to consequences of other first-order axioms. In contrast, the constraint TBox $\mathcal{C}$ has the standard semantics for TBox reasoning and is applicable as usual; it is only for ABox reasoning that $\mathcal{C}$ is applied in a nonstandard way as a check. Thus, the semantics of $\mathcal{C}$ is much closer to the standard semantics of description logics.

Second, the semantics of MKNF makes it difficult to express constraints on unnamed individuals. Namely, for a first-order concept C, the concept $\mathbf{K}\, C$ contains the individuals that are in C in all models of C. In most cases, $\mathbf{K}\, C$ contains only the explicitly named individuals, and not the unnamed individuals implied by existential quantifiers; namely, in different models one can choose different individuals to satisfy an existential quantifier. Therefore, the MKNF-based approaches cannot express the constraints from Section 5.3—that is, they cannot check whether all existentially implied objects are explicitly named.

Third, MKNF-based constraints work at the level of consequences and therefore cannot express constraints on disjunctive facts. Consider again the ABox $\mathcal{A}_3$ containing the axiom (22) and the standard TBox $\mathcal{S}_3$ containing the axiom (23). We might express the constraints (24)–(25) using the following MKNF rules:

$$\leftarrow \mathbf{K}\, Tiger(x), \mathbf{not}\, Carnivore(x) \tag{62}$$

$$\leftarrow \mathbf{K}\, Leopard(x), \mathbf{not}\, Carnivore(x) \tag{63}$$

Unfortunately, (62) and (63) are satisfied for $\mathcal{A}_3 \cup \mathcal{S}_3$. Namely, $\mathbf{K}\, Tiger(x)$ can, roughly speaking, be understood as "$Tiger(x)$ is a consequence." Due to the disjunction in (23), neither $Tiger(ShereKahn)$ nor $Leopard(ShereKahn)$ is a consequence of $\mathcal{A}_3 \cup \mathcal{S}_3$; hence, the premise of neither rule is satisfied and the constraints are not violated.

Because of these differences, we believe that the semantics of the extended DL knowledge bases captures the intuition behind constraints in a much more intuitive way; furthermore, it seems to fit better with the usual semantics of description logics.

8 Conclusion

In this paper we analyzed the similarities and differences between relational databases and description logics, in particular with respect to the role of schema constraints. Our analysis reveals more similarities than differences since, in both cases, constraints are just (restricted) first-order theories. Furthermore, reasoning about the schema is in both systems performed under standard first-order semantics, and it even employs closely related reasoning algorithms. The differences between relational databases and description logics become apparent only if one considers the problems related to reasoning about data. In relational databases, answering queries and constraint satisfaction checking correspond to model checking, whereas, in description logics they correspond to entailment problems.

Based on our analysis, we defined 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; however, for ABox reasoning, they are interpreted in the spirit of relational databases. Hence, our constraints can be used to check whether all necessary assertions have been explicitly specified in an ABox.

We also showed that, if the constraints are satisfied, we can disregard them while answering positive queries. Thus, answering queries under constraints can be computationally easier in these cases.

We presented several procedures for checking constraint satisfaction. For existential-free knowledge bases, constraint checking can be realized using available logic programming engines. If the TBox axioms that are not constraints can be expressed in the DL $\mathcal{ALCHI}$, constraints can be checked by embedding the problem into monadic second-order logic SkS.

For our future work, we shall primarily try to obtain tight complexity bounds for constraint checking in the second case. Furthermore, we shall implement our approach in the DL reasoner KAON2[4] and test its usefulness on practical problems.

References

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

[2] F. Baader, S. Brandt, and C. Lutz. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.

[4] http://kaon2.semanticweb.org/

[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, January 2003.

[4] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In J. Mylopoulos and R. Reiter, editors, *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 452–457, Sydney, Australia, August 24–30 1991. Morgan Kaufmann Publishers.

[5] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From Tableaux to Automata for Description Logics. *Fundamenta Informaticae*, 57:1–33, 2003.

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

[7] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[8] P. Bonatti, C. Lutz, and F. Wolter. Description Logics with Circumscription. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 400–410, Lake District, UK, June 2–5 2006. AAAI Press.

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

[10] A. Borgida and R. J. Brachman. Loading Data into Description Reasoners. *SIGMOD Record*, 22(2):217–226, 1993.

[11] D. Calvanese, D. De Giacomo, and M. Lenzerini. Keys for free in description logics. In F. Baader and U. Sattler, editors, *Proc. of the 2000 Int. Workshop on Description Logic (DL 2000)*, volume 81 of *CEUR Workshop Proceedings*, Aachen, Germany, August 17–19 2000.

[12] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data Complexity of Query Answering in Description Logics. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, Lake District, UK, June 2–5 2006. AAAI Press.

[13] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '98)*, pages 149–158, Seattle, WA, USA, June 1–3 1998. ACM Press.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[15] 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.

[16] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving using the DLV system. *Logic-Based Artificial Intelligence*, pages 79–103, 2000.

[17] M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Texts in Computer Science. Springer, 1996.

[18] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of the 5th Int. Conf. on Logic Programming (ICLP '88)*, pages 1070–1080, Seattler, WA, USA, August 15–19 1988. MIT Press.

[19] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[20] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, Hyderabad, India, January 6–12 2007. Morgan Kaufmann Publishers. To appear.

[21] V. Haarslev and R. Möller. Incremental Query Answering for Implementing Document Retrieval Services. In D. Calvanese, G. de Giacomo, and F. Franconi, editors, *Proc. of the 2003 Int. Workshop on Description Logics (DL 2003)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy, September 5–7 2003.

[22] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Conceptual Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 2006. accepted for publication.

[23] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide Query Containment under Constraints using a Description Logic. In

M. Parigot and A. Voronkov, editors, *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *LNAI*, pages 326–343, Reunion Island, France, November 11–12 2000. Springer.

[24] I. Horrocks and S. Tessaris. Querying the Semantic Web: a Formal Approach. In I. Horrocks and J. A. Hendler, editors, *Proc. of the 1st Int. Semantic Web Conference (ISWC 2002)*, volume 2342 of *LNCS*, pages 177–191, Sardinia, Italy, June 9–12 2002. Springer.

[25] U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.

[26] O. Kutz, I. Horrocks, and U. Sattler. The Even More Irresistible SROIQ. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 68–78, Lake District, UK, June 2–5 2006. AAAI Press.

[27] A. Y. Levy. Obtaining Complete Answers from Incomplete Databases. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of 22th Int. Conf. on Very Large Data Bases (VLDB '96)*, pages 402–412, Mumbai, India, September 3–6 1996. Morgan Kaufmann.

[28] V. Lifschitz. Minimal Belief and Negation as Failure. *Artificial Intelligence*, 70(1–2):53–72, 1994.

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

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

[31] B. Motik and R. Rosati. A Faithful Integration of Description Logics with Logic Programming. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, Hyderabad, India, January 6–12 2007. Morgan Kaufmann Publishers. To appear.

[32] I. Niemeä. A tableau calculus for minimal model reasoning. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proc. of the 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX '96)*, pages 278–294, Terrasini, Italy, May 15–17 1996. Springer.

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

[34] D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Logic and Computation*, 2(3):293–304, 1986.

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

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

[37] R. Rosati. $\mathcal{DL} + log$: A Tight Integration of Description Logics and Disjunctive Datalog. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 68–78, Lake District, UK, June 2–5 2006. AAAI Press.

[38] O. Shmueli. Equivalence of DATALOG Queries is Undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[39] T. Syrjänen and I. Niemelä. The Smodels System. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proc. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *LNAI*, pages 434–438, Vienna, Austria, September 17–19 2001. Springer.

[40] M. Weyer. Decidability of S1S and S2S. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*, pages 207–230. Springer, 2002.