Bridging the Gap Between OWL and Relational Databases*

Boris Motik[†] Ian Horrocks[†]

Ulrike Sattler[‡]

[†]Computing Laboratory,
University of Oxford, UK

[‡]Department of Computer Science,
University of Manchester, UK

March 27, 2009

Abstract

Despite similarities between the Web Ontology Language (OWL) and schema languages traditionally used in relational databases, systems based on these languages exhibit quite different behavior in practice. The schema statements in relational databases are usually interpreted as *integrity constraints* and are used to check whether the data is structured according to the schema. OWL allows for axioms that resemble integrity constraints; however, these axioms are interpreted under the standard first-order semantics and not as checks. This often leads to confusion and is inappropriate in certain data-centric applications. To explain the source of this confusion, in this paper we compare OWL and relational databases w.r.t. their schema languages and basic computational problems. Based on this comparison, we extend OWL with integrity constraints that capture the intuition behind similar statements in relational databases. We show that, if the integrity constraints are satisfied, they need not be considered while answering a broad range of *positive* queries. Finally, we discuss several algorithms for checking integrity constraint satisfaction, each of which is suitable to different types of OWL knowledge bases.

Keywords

Integrity Constraints, Relational Databases, OWL, Semantic Web

*This is an extended version of the paper with the same name published at WWW 2007 [31].

1 Introduction

The Web Ontology Language (OWL) is a W3C standard for modeling ontologies in the Semantic Web. The logical underpinning of OWL is provided by description logics (DLs) [2]. OWL can be seen as an expressive schema language; however, its axioms have a different meaning from similar statements in relational databases. Ontology designers sometimes intend OWL axioms to be read as *integrity constraints* (ICs)—checks that verify whether the information explicitly present in the ontology satisfies certain conditions. The formal semantics of OWL, however, does not interpret these axioms as integrity constraints, so the consequences that one can draw from such ontologies differ from the ones that the users intuitively expect.

To understand the nature of the problem, consider an application for managing tax returns in which each person is required to have a social security number. In a relational database, this would be captured by an inclusion dependency stating that a social security number exists for each person. During database updates, such a dependency is interpreted as a check: whenever a person is added to the database, a check is performed to see whether that person's social security number has been specified as well; if not, the update is rejected. An apparently similar dependency can be expressed in OWL using an existential restriction, but this will result in quite a different behavior: adding a person without a social security number to an OWL knowledge base does not raise an error, but only leads to the inference that the person in question has some (unknown) social security number. OWL is thus closely related to *incomplete* databases [43]—databases whose data is specified only partially.[1] OWL axioms, just like dependencies in incomplete databases, do not check the integrity of the database data; instead, they show how to extend the database with missing information.

In fact, it is not possible to formalize database-like integrity constraints in OWL, which has caused problems in practice. Axioms such as domain and range constraints look like ICs, so users often expect them to behave accordingly, which often leads to problems. On the one hand, such axioms do not check whether the data has been input correctly and, on the other hand, they cause considerable performance overhead during reasoning. These problems could be addressed if OWL were extended with true database-like integrity constraints.

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 allowed states of the

[1] Dependencies in incomplete databases are often called constraints in the literature. To avoid confusion, in this paper we reserve the term "integrity constraint" for axioms that behave as database-style checks.

database, and are therefore of an epistemic nature [38]. 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 [16] or various nonmonotonic rule extensions of DLs [39, 32]. While these approaches do solve the outlined problem to a certain extent, such a solution is not in the spirit of relational databases. As we discuss in more detail in Section 8, integrity constraints in these approaches do not affect TBox reasoning at all, and they are applied only 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.

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 database data, as well as in schema reasoning tasks, such as computing query subsumption. The semantic relationship between these two roles of integrity constraints is much clearer than in autoepistemic ICs, which simplifies modeling.

In order to make schema modeling in OWL more natural for data-centric applications, in this paper we study the relationship between OWL and databases. Based on our analysis, we propose an extension of OWL that mimics the behavior of integrity constraints in relational databases, while keeping the main benefits of OWL such as the capability to model hierarchical domains. The contributions of this paper are as follows.

- In Section 2, we compare OWL and relational databases w.r.t. their schema languages, main reasoning problems, and approaches to modeling integrity constraints. Furthermore, we discuss in detail the relationship between OWL and incomplete databases.

- To allow users to control the degree of incompleteness in OWL, in Section 3 we introduce *extended DL knowledge bases*. The schema part of such knowledge bases is separated into the *standard TBox* that contains axioms which are interpreted as usual, and the *integrity constraint TBox* that contains axioms which are interpreted as checks. We also define an appropriate notion of IC satisfaction based on the notion of minimal models.

- In Section 4, we show that our ICs indeed behave similarly to ICs in relational databases: if the ICs are satisfied in an extended DL knowledge base, they need not be considered while answering a broad class of positive ABox queries. This result promises a significant performance improvement of query answering in practice, as it allows us to consider a subset of the TBox during query answering.

- In Section 5, we show how to incorporate the modeling of ICs into the general ontology modeling process, and we also discuss the types of axiom that are likely to be designated as integrity constraints.
- In Section 6, we present an alternative characterization of IC satisfaction by embedding the problem into logic programming. This provides us with additional intuition behind the notion of IC satisfaction, and it also lays the foundation for a practical decision procedure.
- In Section 7, we present several algorithms for checking IC satisfaction in different types of knowledge bases. For knowledge bases without positively occurring existential quantifiers, IC satisfaction can be checked using existing logic programming machinery. For knowledge bases with existential quantifiers, we embed the IC satisfaction problem into the monadic second-order logic on infinite k-ary trees *SkS* [36]. We do not expect this procedure to be practical; rather, it merely shows us that IC satisfaction is decidable, and that a more practical procedure might exist.
- In Section 8, we discuss how our approach relates to the existing approaches for modeling integrity constraints.

We assume the reader to be familiar with the basics of OWL and DLs; please refer to [2] for an introduction. It is well-known that the OWL DL variant of OWL corresponds to the DL $\mathcal{SHOIN}(\mathbf{D})$. Because of that, we refer to OWL and DLs interchangeably throughout this paper.

2 OWL vs. Relational Databases

An obvious distinction between OWL/DLs and relational databases is that the former use open-world semantics, whereas the latter use closed-word semantics. We argue that the two semantics actually complement each other and that the choice of the semantics should depend on the inference problem.

2.1 Schema Language

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

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

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

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

Most DLs can be seen as decidable fragments of first-order logic [8], so axioms (1)–(3) can be translated into the following first-order formulae:

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

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

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

The schema of a relational database is defined in terms of relations and dependencies. Many types of dependencies have been considered in the literature, such as functional, inclusion, and join dependencies. As discussed in [1], most dependencies can be represented as first-order formulae of the form (7), where ψ and ξ are conjunctions of function-free atoms:

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

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

2.2 Interpreting the Schema

DL TBoxes and relational schemata are interpreted according to the standard first-order semantics: they distinguish the legal from the illegal relational structures—that is, the structures that satisfy all axioms from the structures that violate some axiom. In DLs, the legal structures are called *models*, whereas in relational databases they are called *database instances*, but the underlying principle is the same.

There is a slight technical difference between models and database instances: models can be infinite, whereas database instances are typically required to be finite since only finite databases can be stored in practical systems. For many classes of dependencies, whenever an infinite relational structure satisfying the schema exists, a finite structure exists as well (this is known as the *finite model* property), so the restriction to finite structures is not really relevant. Languages such as OWL do not have the finite model property: ontologies exists that are satisfied only in infinite models [2]. Even though the complexity of finite model reasoning is, for numerous DLs, the same as the complexity of reasoning w.r.t. arbitrary models, the former is usually more involved [30, 35]. Hence, in the rest of this paper, we drop the restriction to finite database instances and consider models and database instances to be synonymous.

2.3 Domains and Typing

Relational databases assign domain types to columns of relations; for example, the second position in the *hasSSN* relation could be restricted to strings of a certain form. Typing is used in practice to determine the physical layout of the database. In contrast, typing is often not considered in theory (e.g., in algorithms for checking query containment); rather, all columns are assumed to draw their values from a common countable domain set [1].

In DLs underlying OWL, *datatypes*—a simplified variant of *concrete domains* [3]—can be used to specify types of data.

In this paper, we consider neither typed relational schemata nor DL knowledge bases with datatypes, and we interpret both in standard first-order logic. This simplifies both formalisms significantly. For example, keys can be straightforwardly added to untyped DLs [11], while adding keys to DLs with datatypes is much more involved [29].

2.4 Schema Reasoning

Checking subsumption relationships between concepts has always been a central reasoning problem for DLs. A concept C is *subsumed* by a concept D w.r.t. a DL TBox $\mathcal{T}$ if the extension of C is included in the extension of D in each model I of $\mathcal{T}$. This inference has many uses; for example, in ontology modeling, derived subsumption relationships can be used to detect modeling errors. Concept subsumption has been used to optimize query answering [19], especially when generalized to subsumption of conjunctive queries [12, 18]. Another important TBox inference is checking concept satisfiability—that is, determining whether a model of $\mathcal{T}$ exists in which a given concept has a nonempty extension. Concepts are unsatisfiable mainly due to modeling errors, so this inference can also be used to detect ontology modeling errors.

Reasoning about the schema is certainly not the most prominent feature of relational databases, yet a significant amount of research has been devoted to it. The most important schema-related inference in databases is checking *query containment* [1]: a query Q_1 is contained in a query Q_2 w.r.t. a schema $\mathcal{T}$ if the answer to Q_1 is contained in the answer to Q_2 for each database instance that satisfies $\mathcal{T}$. This inference is used by database systems to rewrite queries into equivalent ones that can be answered more efficiently. Another useful schema reasoning problem is *dependency minimization*—that is, computing a minimal schema that is equivalent to the given one.

In both DLs and relational databases, schema reasoning problems correspond to checking whether some formula φ holds in every model (i.e., database instance) of $\mathcal{T}$—that is, checking whether $\mathcal{T} \models \varphi$. In other words, the schema reasoning problems in both DLs and relational databases correspond to *entailment* in 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: reasoning in DLs is typically performed by tableau algorithms [4], and the state-of-the-art reasoning technique in relational databases is chase [1]. Apart from notational differences, the principles underlying these two techniques are the same: they both try to construct a model that satisfies the schema $\mathcal{T}$ but not the formula φ.

To summarize, DLs and databases treat schema reasoning problems in the same way. Thus, DLs can be understood as expressive but decidable (database) schema languages.

2.5 Query Answering

Apart from the schema (or TBox) part, a DL knowledge base $\mathcal{K}$ typically also has a data (or ABox) part. The main inference for ABoxes is *instance checking*—that is, checking whether an individual a is contained in the extension of a concept C in every model of $\mathcal{K}$, commonly written as $\mathcal{K} \models C(a)$. Instance checking can be generalized to answering conjunctive queries over DL knowledge bases [12, 18], so a DL query can be viewed as a first-order formula φ with free variables $x_1, \ldots, x_n$. Just like schema reasoning, the semantics of query answering in DLs is defined as first-order entailment, so it takes into account all models of $\mathcal{K}$: a tuple $a_1, \ldots, a_n$ is an *answer* to φ over $\mathcal{K}$ if $\mathcal{K} \models \varphi[a_1/x_1, \ldots, a_n/x_n]$, where the latter formula is obtained from φ by replacing all free occurrences of x_i with a_i.

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 DLs. A significant difference between DLs and relational databases is, however, the way in which queries are evaluated. Let φ be a first-order formula with 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 DLs, query answering in relational databases does not consider all databases instances that satisfy the knowledge base $\mathcal{K}$; instead, it considers only the given instance I. In other words, query answering in relational databases is not defined as entailment, but as model checking, where the model is the given database instance.

Although the definition of query answering in relational databases from the previous paragraph is the most widely used one, a significant amount of research has also been devoted to answering queries over incomplete databases [25, 21, 43]—a problem that is particularly interesting in information integration. An incomplete database $\mathcal{DB}$ is described by a set $\mathcal{R}$ of incomplete extensions of the schema relations and a set $\mathcal{S}$ of dependencies specifying how the incomplete extensions relate to the actual database instance. Queries in incomplete databases are also (possibly restricted) first-order formulae. In contrast to complete databases, a tuple $a_1, \ldots, a_n$ is a *certain* answer to φ over $\mathcal{DB}$ if $I \models \varphi[a_1/x_1, \ldots, a_n/x_n]$ for each database

instance I that satisfies $\mathcal{R}$ and $\mathcal{S}$. In other words, query answering in incomplete databases is defined as first-order entailment just like in DLs, where the relation extensions correspond to the DL ABox and the schema corresponds to the DL TBox. Hence, from the standpoint of query answering, DLs can be understood as incomplete databases.

2.6 Checking Satisfaction of Integrity Constraints

Integrity constraints play a central role in relational databases, where they are used to ensure data integrity. We explain the intuition behind ICs by means of an example. Let $\mathcal{T}$ be a relational schema containing the statement (4), and let I be a database instance containing only the following fact:

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

To check whether all data has been specified correctly, we can now ask whether the ICs in $\mathcal{T}$ are satisfied for I; that is, whether $I \models \mathcal{T}$. In our example, this is not the case: integrity constraint (4) says that each database instance must contain an SSN for each person. Since I does not contain the SSN of *Peter*, the ICs in $\mathcal{T}$ are not satisfied.[2] The database instance is fully specified by the facts available in the database; hence, all data is assumed to be complete. Thus, IC satisfaction checking in relational databases is based on model checking.

In DLs, we can check whether an ABox $\mathcal{A}$ is consistent with a TBox $\mathcal{T}$—that is, whether a model I of both $\mathcal{A}$ and $\mathcal{T}$ exists—and thus detect possible contradictions in $\mathcal{A}$ and $\mathcal{T}$. This inference, however, does not provide us with a suitable basis for IC satisfaction checking. For example, let $\mathcal{T}$ and $\mathcal{A}$ contain axiom (1) and fact (8), respectively. The knowledge base $\mathcal{A} \cup \mathcal{T}$ is satisfiable: axiom (1) is not interpreted as a check, but it implies that *Peter* has some (unknown) SSN. This clearly does not match with our intuition behind integrity constraints. First-order satisfiability checking verifies whether the facts in $\mathcal{A}$ can be extended to a relational structure that is compatible with the schema $\mathcal{T}$, thus assuming that our knowledge about the world is incomplete. To the best of our knowledge, no description logic currently provides an inference that matches with the intuition behind database-like integrity constraints.

2.7 Discussion

From the standpoint of conceptual modeling, DLs 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 determines the legal database instances. In fact, when computing

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

the subsumption relationship between concepts or queries, we do not have a fixed instance. Therefore, we cannot interpret the schema in either OWL or relational databases under the closed-world assumption; rather, we must employ the open-world semantics in order to consider all instances.

Integrity constraints are mostly useful in data-centric applications—that is, applications that focus on the management of large volumes of data. In practice, relational databases are typically complete: any missing information is either encoded metalogically (e.g., users often include fields such as *hasSpecifiedSSN* to signal that particular data has been supplied in the database), or it is represented by *null-values* (that can be given first-order interpretation [21]). In contrast, ABoxes in DLs are closely related to incomplete (relational) databases. Clearly, problems may arise if certain aspects of the information about individuals in ABoxes are expected to be complete. To understand the problems that occur in such cases, consider the following example taken from the Biopax[3] ontology used for data exchange between biological databases. This ontology defines the domain of the property *NAME* to be the union of *bioSource*, *entity*, and *dataSource*:

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

The intention behind this axiom is to define which objects can be named—that is, to ensure that a name is attached only to objects of the appropriate type. The actual data in the Biopax ontology is complete w.r.t. this integrity constraint: each object with a name is also typed (sometimes indirectly through the class hierarchy) to at least one of the required classes. Axiom (9) is, however, not interpreted as an integrity constraint in OWL; rather, this axiom says that, if some object has a name, then it can be *inferred* to be either a *bioSource*, an *entity*, or a *dataSource*. Therefore, axiom (9) cannot be used to check whether all data is correctly typed. Furthermore, since axiom (9) contains a disjunction in the consequent, an OWL reasoner processing the Biopax ontology must use reasoning-by-case, which is one of the reasons why DL reasoning is intractable [2, Chapter 3]. Hence, axiom (9) causes two types of problems: on the one hand, it does not exhibit the intended behavior and, on the other hand, it introduces a performance penalty during reasoning.

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

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

To draw an inference using this axiom, we do not necessarily need to know the name of the spouse; we only need to know that a spouse exists. Thus,

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

we may state the following fact:

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

We are now able to derive that *Peter* is eligible for a tax cut even without knowing the name of his spouse. Providing complete information can be understood as filling in a "Spouse name" box on a tax return, whereas providing incomplete information can be understood as just ticking the "Married" box. The existential quantifier can be understood as a well-behaved version of null-values that explicitly specifies the semantics of data incompleteness [21]. Thus, DLs provide a sound and well-understood foundation for use cases that require reasoning with incomplete information.

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

3 Integrity Constraints for OWL

In this section, we extend DL knowledge bases with ICs in order to overcome the problems discussed in the previous section. Since TBoxes are first-order formulae, it is straightforward to apply the model checking approach described in Section 2 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, and the ICs would be satisfied if $\mathcal{A} \models \mathcal{T}$. Such an approach is, however, not satisfactory as it requires an "all-or-nothing" choice: the ABox is then considered to be a complete model, and TBox axioms can only be used as checks and not to imply new facts.

To obtain a more versatile formalism, we propose a combination of inferencing and model checking. The following example demonstrates the desirable behavior of our approach. Let $\mathcal{A}_1$ be the following ABox:

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

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

$$Student \sqsubseteq Person \tag{16}$$

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

Let us now assume that we choose (17) to be an integrity constraint, but (16) to be a normal axiom. Since (16) is a normal axiom, we should derive $Person(Peter)$ and $Person(Paul)$. Axiom (17) is an IC, so it should be applied as a check. Hence, we expect the IC to be satisfied for *Peter* since an SSN for *Peter* has been specified; furthermore, no SSN has been specified for *Paul*, so we expect IC (17) to be violated for *Paul*.

Following this intuition, we define extended DL knowledge bases to distinguish the axioms that imply new facts (i.e., the axioms that act as "deductive rules") from the integrity constraints (i.e., the axioms that act as checks). It is important to understand that, whether an axiom is to be treated as an IC or not, depends mainly on the requirements and the assumptions of the application and is often not inherent in a particular axiom. In fact, given the same ontology, different applications might choose to interpret different subsets of the ontology's axioms as ICs. Thus, it might be beneficial to keep the information about which axioms are treated as ICs independently from the core ontology, thus allowing applications to create application-specific views of the ontology. A discussion of such a mechanism, however, is out of scope of this paper; here, we focus on the semantic and computational aspects of integrity constraints.

The following definition is applicable to any DL, so we do not formally introduce here a particular description logic; please refer to related literature [2] for a formal definition of the DLs used in the Semantic Web. Furthermore, our definition allows ABoxes to contain only possibly negated atomic concepts. This does not result in any loss of generality because $\mathcal{S}$ can be used to introduce names for nonatomic concepts.

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

In the rest of this section, we explore the possible notions of IC satisfaction. Several variants have already been considered in the literature.

In the *consistency* approach for *open databases* [24], ICs are satisfied if $\mathcal{A} \cup \mathcal{S}$ is consistent with $\mathcal{C}$—that is, if $\mathcal{A} \cup \mathcal{S} \cup \mathcal{C}$ is satisfiable. Reiter, however, has already observed that such a treatment of ICs is too weak [38]: $\mathcal{C}$ is then treated as a standard first-order theory, so, as explained in Section 2.6, ICs do not behave as checks.

In the *entailment* approach for *open databases* [37], ICs are satisfied if they are true in each model of $\mathcal{A} \cup \mathcal{S}$—that is, if $\mathcal{A} \cup \mathcal{S} \models \mathcal{C}$. The following example, similar to the one by Reiter [38], shows that this does not satisfy our intuition. Let $\mathcal{A}_2$ contain only fact (12), let $\mathcal{S}_2 = \emptyset$, and let $\mathcal{C}_2$ contain only axiom (17). The interpretation $I = \{Student(Peter), Person(Peter)\}$ is a model of $\mathcal{A}_2 \cup \mathcal{S}_2$ that does not satisfy $\mathcal{C}_2$, which would make $\mathcal{C}_2$ not satisfied for $\mathcal{A}_2 \cup \mathcal{S}_2$. Intuitively, though, the fact $Person(Peter)$ is not implied by $\mathcal{A}_2 \cup \mathcal{S}_2$, so we should not check whether *Peter* has an SSN at all. We want $\mathcal{C}_2$ to hold only for the facts that are implied by $\mathcal{A}_2 \cup \mathcal{S}_2$.

Consistency and entailment approaches were applied to closed databases (see [40, 33] for consistency and [27] for entailment), where ICs are satisfied if the Clark's completion [14] of $\mathcal{A} \cup \mathcal{S}$ is consistent with (resp. it entails) $\mathcal{S}$. Such approaches, however, are applicable only to Prolog-like databases for which Clark's completion is defined—that is, they are not applicable to databases containing disjunction or existential quantification [38]—and are therefore not applicable to most description logics.

The example from the discussion of the entailment approach in open databases suggests yet another definition of IC satisfaction: $\mathcal{C}$ should hold for all first-order consequences of $\mathcal{A} \cup \mathcal{S}$. On $\mathcal{A}_2$, $\mathcal{C}_2$, and $\mathcal{S}_2$ this produces the desired behavior: $Person(Peter)$ is not a consequence of $\mathcal{A}_2 \cup \mathcal{S}_2$, so the integrity constraint from $\mathcal{C}_2$ should not be checked for *Peter*. Consider, however, the ABox $\mathcal{A}_3$ containing only the following fact:

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

Furthermore, let $\mathcal{S}_3$ contain the following axiom:

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

Finally, let $\mathcal{C}_3$ contain the following two integrity constraints:

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

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

Neither $Tiger(ShereKahn)$ nor $Leopard(ShereKahn)$ is a first-order consequence of $\mathcal{A}_3 \cup \mathcal{S}_3$, which means that the ICs in $\mathcal{C}_3$ are satisfied; furthermore, $\mathcal{A}_3 \cup \mathcal{S}_3 \not\models Carnivore(ShereKahn)$. This does not satisfy our intuition: in each model of $\mathcal{A}_3 \cup \mathcal{S}_3$, either the fact $Tiger(ShereKahn)$ or the fact $Leopard(ShereKahn)$ is true, but the fact $Carnivore(ShereKahn)$ is not necessarily true in either case. Hence, by treating (20)–(21) as integrity constraints and not as standard axioms, we neither get an IC violation nor derive the consequence $Carnivore(ShereKahn)$.

Intuitively, integrity constraints should check whether the facts derivable from $\mathcal{A} \cup \mathcal{S} \cup \mathcal{C}$ are also derivable using $\mathcal{A} \cup \mathcal{S}$ only. This notion seems to be nicely captured by minimal models; hence, we check $\mathcal{C}$ only w.r.t. 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 $I' \subsetneq I$ is not a model of φ, where we consider an interpretation to be equivalent to the set of positive ground facts that are true in the interpretation. 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 integrity constraint (17) is not violated. In contrast, $\mathcal{A}_3 \cup \mathcal{S}_3$ has exactly two minimal models:

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

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

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

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

$$\begin{aligned} I'_1 &= I_1 \cup \{Carnivore(ShereKahn)\} \\ I'_2 &= I_2 \cup \{Carnivore(ShereKahn)\} \end{aligned}$$

Both I'_1 and I'_2 satisfy $\mathcal{C}_4$, which matches our intuition. Also, observe that $\mathcal{A}_4 \cup \mathcal{S}_4 \models Carnivore(ShereKahn)$; hence, we derive exactly the same consequences, even though we treat (20)–(21) as ICs.

Minimal models have been used, with minor differences, in an extension of DLs with circumscription [7] and in the semantics of open answer set programs [20]. These well-known definitions, however, seem inappropriate for the definition of IC satisfaction. Consider the following ABox $\mathcal{A}_5$:

$$Woman(Alice) \tag{23}$$

$$Man(Bob) \tag{24}$$

Furthermore, let $\mathcal{S}_5 = \emptyset$ and let $\mathcal{C}_5$ contain the following integrity constraint:

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

No axiom implies that *Alice* and *Bob* are the same, so we expect them to be different "by default", thus making integrity constraint (25) satisfied. The definitions from [7, 20], however, 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, however, is rather restrictive and is not compatible with OWL, which does not employ the UNA. Another solution is to interpret $\mathcal{A} \cup \mathcal{S}$ in a Herbrand model (i.e., a model in which each constant is interpreted by itself) where $\approx$ is a congruence relation; then, we minimize the interpretation of $\approx$ together with all the other predicates. In such a case, the only minimal model of $\mathcal{A}_5$ is $I' = \{ Woman(Alice), Man(Bob) \}$ since the extension of $\approx$ is empty due to minimization, so $\mathcal{C}_5$ is satisfied in I'.

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

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

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

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

Finally, let $\mathcal{C}_6$ contain the following integrity constraint:

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

IC (31) is satisfied in $\mathcal{A} \cup \mathcal{S}_6$, which might be considered intuitive: no axiom in $\mathcal{S}_6$ forces the children of *Peter*—the two individuals whose existence is implied by (29) and (30)—to be the same, so we might conclude that they are different.

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

$$HasChild(Peter) \tag{32}$$
$$TwoChildren(Peter) \tag{33}$$

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

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

If we follow the intuition from the previous example, then $\mathcal{C}_7$ should be satisfied in $\mathcal{A}_7 \cup \mathcal{S}_7$: no axiom in $\mathcal{S}_7$ makes the two individuals introduced by axiom (34) the same, so we can assume that these individuals are different. In contrast, let $\mathcal{S}'_7$ be a standard TBox containing only the following axiom:

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

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

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

The example involving $\mathcal{S}_7$ and $\mathcal{S}'_7$ reveals an important disadvantage of the first reading: if we require each existential quantifier to introduce a distinct individual, then it is possible for an IC 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 IC satisfaction to depend on the syntactic structure of the standard TBox.

The introduction of distinct individuals for each existential quantifier can be justified, however, by *Skolemization* [34], the well-known process of representing existential quantifiers with new function symbols. For example, the Skolemization of the formula $\varphi = \exists y : [R(x, y) \wedge C(y)]$ produces the formula $\mathsf{sk}(\varphi) = R(x, f(x)) \wedge C(f(x))$: the variable y is replaced by a term $f(x)$ with f a fresh function symbol. Skolemized formulae are usually interpreted in *Herbrand* models, whose domain consists of all ground terms built from constants and function symbols in the formula. If a formula contains at least one function symbol, then its Herbrand models are infinite; furthermore, the models of DL axioms are forest-like (i.e., they can be viewed as trees possibly interconnected at roots). We use these properties in our procedure for checking IC satisfaction that we present in Section 7.

Definition 2. *Let φ be a first-order formula and $\mathsf{sk}(\varphi)$ the formula obtained by outer Skolemization of φ [34]. 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' w.r.t. φ such that $I' \subsetneq I$. 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 IC satisfaction, which is based on a translation into first-order logic. For a set of DL axioms S, with $\pi(S)$ we denote the first-order formula with equality and counting quantifiers that is equivalent to S. Such translations are well known for most DLs [2, 8].

Definition 3. *Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base. The integrity 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 $\mathsf{sk}(\mathcal{A} \cup \mathcal{S}) \models_{\mathsf{MM}} \mathcal{C}$.*

The addition of ICs does not change the semantics of DLs or OWL: Definition 3 is only concerned with the semantics of ICs, and an ordinary DL knowledge base $(\mathcal{T}, \mathcal{A})$ can be seen as an extended knowledge base $(\mathcal{T}, \emptyset, \mathcal{A})$. For subsumption and concept satisfiability tests, we should use $\mathcal{S} \cup \mathcal{C}$ together as one common schema, just as it is the case in standard DLs. All inference problems are defined as usual; for example, a concept C is subsumed by a concept D if the extension of C is included in the extension of D in every model of $\mathcal{S} \cup \mathcal{C}$.

Integrity constraints become important only in combination with an ABox $\mathcal{A}$. We invite the reader to verify that, on the examples presented thus far, Definition 3 indeed provides a semantics for ICs that follows the principles from relational databases. In Section 4 we show that, if ICs are satisfied, we can throw them away without losing any positive consequences; that is, we can answer positive queries by taking into account only $\mathcal{A}$ and $\mathcal{S}$. This further shows that our ICs are similar to the integrity constraints in relational databases.

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

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

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

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

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

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

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

When written in the latter form, it can be seen that the IC should be satisfied, since $\mathit{Meaty}(\mathit{soup})$ is not derivable.

As this example illustrates, the intuitive meaning of integrity constraints is easier to grasp if we transform them into the form $C \sqsubseteq D$, where both C and D are negation-free concepts. This is because, by Definition 3, our ICs check only 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:

$$\mathit{NotMeaty} \equiv \neg \mathit{Meaty} \tag{40}$$

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

$$\mathit{Vegetarian} \sqsubseteq \forall \mathit{eats}.\mathit{NotMeaty} \tag{41}$$

IC (41) is now of the "positive" 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 *NotMeaty*. Now $\mathcal{A}_9 \cup \mathcal{S}_9$ has the following two minimal models, and $I_3 \not\models \mathcal{C}_9$, so $\mathcal{C}_9$ is not satisfied in $\mathcal{A}_9$ and $\mathcal{S}_9$:

$$\begin{aligned} I_3 &= \{\mathit{Vegetarian}(\mathit{Ian}), \mathit{eats}(\mathit{Ian}, \mathit{soup}), \mathit{Meaty}(\mathit{soup})\} \\ I_4 &= \{\mathit{Vegetarian}(\mathit{Ian}), \mathit{eats}(\mathit{Ian}, \mathit{soup}), \mathit{NotMeaty}(\mathit{soup})\} \end{aligned}$$

If we add to $\mathcal{A}_9$ the fact $\mathit{NotMeaty}(\mathit{soup})$, then only I_4 is a minimal model, and $\mathcal{C}_9$ becomes satisfied as expected. Hence, it may be advisable to restrict ICs to positive formulae in order to avoid such misunderstandings.

4 Integrity Constraints and Queries

We now present an important result about answering unions of positive conjunctive queries in extended DL knowledge bases: if the ICs are satisfied, we need not consider them in query answering. This suggests that our semantics of IC satisfaction is reasonable: ICs are checks and, if they are satisfied, we can disregard them without losing relevant consequences. This result is practically important because it simplifies query answering. In Section 7 we show that, for certain types of OWL ontologies, both checking IC satisfaction and query answering can be easier than standard DL reasoning. Note that ICs can still be useful for semantic query optimization—that is, they can be used to reformulate a query into one that can be evaluated more efficiently over all databases that satisfy the ICs (see, e.g., [13, 6, 22]). A discussion of semantic query optimization is, however, beyond the scope of this paper; our main concern here is to show that, if the ICs are satisfied, they are redundant from the semantic point of view.

Before proceeding, we first remind the reader of the definition of unions of conjunctive queries over DL knowledge bases [12].

Definition 4. *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 each t_i is either a constant, a distinguished, or a nondistinguished variable.[4] *A* union of n conjunctive queries *is the formula* $U(\mathbf{x}) = \bigvee_{i=1}^{n} \exists \mathbf{y_i} : Q_i(\mathbf{x}, \mathbf{y_i})$. *A tuple of constants* $\mathbf{c}$ *is an* answer *to* $U(\mathbf{x})$ *over a DL knowledge base* $\mathcal{K}$, *written* $\mathcal{K} \models U(\mathbf{c})$, *if* $\pi(\mathcal{K}) \models U(\mathbf{x})[\mathbf{c}/\mathbf{x}]$.

We first prove an auxiliary lemma.

Lemma 1. *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 $I \subseteq I'$.*

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 [5]. Let I' be a Herbrand model of $\mathsf{sk}(\varphi)$, and let

$$S = \{\mathsf{sk}(\varphi)\} \cup \{\neg A \mid A \text{ is a ground fact over the signature of } \mathsf{sk}(\varphi) \text{ and } A \notin I'\}.$$

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 $\mathsf{sk}(\varphi)$. □

The main result of this section is captured by the following theorem:

Theorem 1. *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 $U(\mathbf{x})$ over $\mathcal{K}$ and any tuple of constants $\mathbf{c}$, we have $\mathcal{A} \cup \mathcal{S} \cup \mathcal{C} \models U(\mathbf{c})$ if and only if $\mathcal{A} \cup \mathcal{S} \models U(\mathbf{c})$.*

Proof. We show the contrapositive: if $\mathcal{K}$ satisfies $\mathcal{C}$, then $\mathcal{S} \cup \mathcal{A} \cup \mathcal{C} \not\models U(\mathbf{c})$ if and only if $\mathcal{S} \cup \mathcal{A} \not\models U(\mathbf{c})$. The ($\Rightarrow$) direction holds trivially, so we consider the ($\Leftarrow$) direction. If $\mathcal{S} \cup \mathcal{A} \not\models U(\mathbf{c})$, then $\mathsf{sk}(\mathcal{S} \cup \mathcal{A} \cup \{\neg U(\mathbf{c})\})$ is satisfiable in a Herbrand model I'. The formula $\neg U(\mathbf{c})$ 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: $\mathsf{sk}(\mathcal{S} \cup \mathcal{A} \cup \{\neg U(\mathbf{c})\}) = \mathsf{sk}(\mathcal{S} \cup \mathcal{A}) \cup \{\neg U(\mathbf{c})\}$. By Lemma 1, a minimal Herbrand interpretation $I \subseteq I'$ exists such that $I \models \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$. Now $I' \models \neg U(\mathbf{c})$, so $I' \models \forall \mathbf{y_i} : \neg Q_i(\mathbf{c}, \mathbf{y_i})$ for each $1 \leq i \leq n$. Hence, for each tuple $\mathbf{t}$ of the elements of the Herbrand universe, $I' \models \neg Q_i(\mathbf{c}, \mathbf{y_i})[\mathbf{t}/\mathbf{y_i}]$. But then, since $I \subseteq I'$ and all atoms from $Q_i(\mathbf{c}, \mathbf{y_i})$ are positive, we have $I \models \neg Q_i(\mathbf{c}, \mathbf{y_i})[\mathbf{t}/\mathbf{y_i}]$ for each $\mathbf{t}$ as well, so $I \models \forall \mathbf{y_i} : \neg Q_i(\mathbf{c}, \mathbf{y_i})$, and therefore $I \not\models \exists \mathbf{y_i} : Q_i(\mathbf{c}, \mathbf{y_i})$. Thus, we conclude $I \not\models U(\mathbf{c})$. Since the ICs are satisfied in $\mathcal{K}$, the axioms in $\mathcal{C}$ are 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 $I \not\models U(\mathbf{c})$. □

[4] The predicate A can be the equality predicate $\approx$, an atomic concept, a role, or an n-ary predicate in case of n-ary DLs.

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

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

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

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

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

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

Under the standard semantics, $\mathcal{K}$ implies the following conclusion:

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

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

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

Note that both entailments in Theorem 1 use the standard semantics of DLs; that is, we do not assume a closed-world semantics for query answering. Furthermore, Theorem 1 does not guarantee preservation of negative consequences; in fact, such consequences may change, as the following example demonstrates. Let $\mathcal{S}_{11} = \emptyset$, let $\mathcal{C}_{11}$ contain the integrity constraint

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

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

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

Furthermore, integrity constraint (48) is satisfied in $\mathcal{S}_{11} \cup \mathcal{A}_{11}$; however, if we disregard $\mathcal{C}_{11}$, we lose the above consequence:

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

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

The proof of Theorem 1 reveals why $U(\mathbf{x})$ is restricted to positive atoms. Consider a model I' such that $I' \models \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$ and $I' \not\models \neg A(a)$. For a minimal model I of $\mathsf{sk}(\mathcal{S} \cup \mathcal{A})$, it might be that $A(a) \in I' \setminus I$, so $I \models \neg A(a)$. Intuitively, IC satisfaction ensures that all positive atoms derivable through ICs are derivable without ICs as well; this, however, does not necessarily hold for negated atoms. This proof also reveals why the entailment of universally quantified formulae is not preserved. Intuitively, for such formulae, we should not consider only the Herbrand models of $\mathsf{sk}(\mathcal{S} \cup \mathcal{A})$ because they may be "too small." For example, let $\mathcal{A} = \{A(a)\}$ and $U = \forall x : A(x)$. Clearly, $\mathcal{A} \not\models U$, but the only Herbrand model of $\mathcal{A}$ is $I = \{A(a)\}$ and $I \models U$. The problem is that I does not take into account the individual that would be introduced by negating and Skolemizing the query.

Theorem 1 has an important implication with respect to TBox reasoning. Let $U_1(\mathbf{x})$ and $U_2(\mathbf{x})$ be unions of conjunctive queries such that $\pi(\mathcal{K}) \models \forall \mathbf{x} : [U_1(\mathbf{x}) \rightarrow U_2(\mathbf{x})]$. Provided that $\mathcal{C}$ is satisfied in $\mathcal{K}$, each answer to $U_1(\mathbf{x})$ w.r.t. $\mathcal{A} \cup \mathcal{S}$ is also an answer to $U_2(\mathbf{x})$ w.r.t. $\mathcal{A} \cup \mathcal{S}$. In other words, we can check subsumption of unions of conjunctive as usual, by treating $\mathcal{C} \cup \mathcal{S}$ as an ordinary DL TBox; subsequently, for knowledge bases that satisfy $\mathcal{C}$, we can ignore $\mathcal{C}$ when answering queries, but query answers will still satisfy the established subsumption relationships between queries.

We conclude this section with the remark that, if an extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ does not satisfy $\mathcal{C}$, then the ICs in $\mathcal{C}$ cannot be ignored during query answering. In such a case it might not actually make sense to attempt to answer any queries over $\mathcal{K}$—that is, IC satisfaction should be taken as a prerequisite for query answering. This is similar to the situation in relational databases systems, which typically reject database updates that violate one or more integrity constraints; thus, ICs are satisfied at all times, which is taken as a precondition for query answering.

5 Using Integrity Constraints in Practice

To clarify our ideas and provide practical guidance, we present in Figure 1 a sequence of steps that, we believe, could be followed to integrate ICs into ontology-based applications. In the rest of this section we consider different steps in more detail.

5.1 Modeling the Domain

The difference between ICs and standard axioms plays no role during domain modeling; that is, the domain should be modeled as usual. For example, to describe that each person should have a social security number, we should simply state axiom (1), and we should not worry at this point whether this axioms should be placed into the standard or the integrity constraint

TBox. Since no data is available during domain modeling, the ontology is modeled as usual. Hence, we classify the knowledge base and check concept satisfiability using well-known tools and techniques.

5.2 Identifying Integrity Constraints

The axioms in both the standard and the integrity constraint TBox describe the general properties of modeled domain; for example, axiom (1) states that each person must have an SSN. In addition to describing the domain, ICs also describe the admissible states of the knowledge base—that is, they describe the assumptions that applications make about the data. It makes sense to consider the application's assumptions about the data separately from domain modeling; hence, we should model a knowledge base first and then subsequently identify certain axioms as integrity constraints.[5] For example, if an application requires the SSN of each person to be known explicitly, then (1) should be moved into the integrity constraint TBox $\mathcal{C}$; otherwise, it should be kept in the standard TBox $\mathcal{S}$. In the rest of this section, we discuss three kinds of axioms that are likely to be identified as ICs.

Participation 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; often, they also define the cardinality of the relationship. The general form of such constraints is as follows, where $\bowtie \in \{\leq, \geq, =\}$ and n is a nonnegative integer:

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

Participation constraints are closely related to inclusion dependencies from relational databases.

Axiom (1) is a typical participation constraint. Another example is the following statement, which allows each person to have at most one spouse:

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

[5] In practice, domain modeling might be interleaved with IC modeling; however, it is beneficial to separate the two steps at least conceptually.

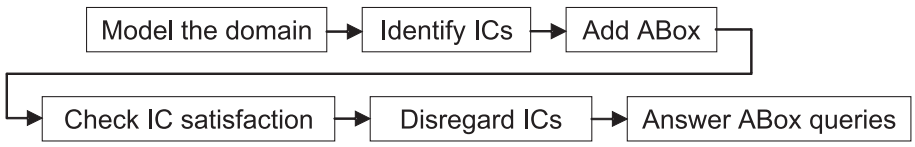

Figure 1: Using Knowledge Bases with Integrity Constraints

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

$$Person(Peter) \tag{51}$$
$$marriedTo(Peter, Ann) \tag{52}$$
$$marriedTo(Peter, Mary) \tag{53}$$

If we treat (50) as a standard TBox axiom (i.e., as part of $\mathcal{S}$), then $\mathcal{A} \cup \mathcal{S}$ is satisfiable; furthermore, due to (50), we derive $Ann \approx Mary$. If, however, we identify (50) as an integrity constraint, then the only minimal model of $\mathcal{A}$ contains exactly facts (51)–(53). This is because the equality predicate $\approx$ is minimized as well, so *Ann* is different from *Mary*. This matches our intuition, as there is no axiom in $\mathcal{S}$ that would force *Ann* and *Mary* to be the same. Thus, *Peter* is married to two different people, so integrity constraint (50) is not satisfied in $\mathcal{A}$.

Typing constraints can be used to check whether objects are correctly typed. Typical examples of such statements are domain and range restrictions: when interpreted as integrity 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. Domain and range constraints are generally of the forms (54) and (55), respectively.

$$\exists R.\top \sqsubseteq C \tag{54}$$
$$\top \sqsubseteq \forall R.C \tag{55}$$

Axiom (9) is a typical example of a domain constraint. Another example is the following axiom, which states that the *marriedTo* relation can point only to instances of the *Person* class:

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

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

Naming constraints can be used to check whether 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. In such an application, we might use axioms (1)–(3) not as ICs, but as elements of the standard TBox $\mathcal{S}$. Furthermore, to distinguish people who have submitted a tax return, we would introduce a concept *PersonTR* for such persons and make it a subset of *Person* in $\mathcal{S}$:

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

Two things should hold for each instance of *PersonTR*: first, we require each such person to be explicitly known by name, and second, we require the SSN of each such person to be known by name as well. Although ICs 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 integrity constraints:

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

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

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

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

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

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

For TBox reasoning, facts of the form (60) are, by definition, not taken into account. Instead of these facts, one might be tempted to use the following axiom, where a_i are all individuals from the ABox:

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

This, however, requires nominals in the DL language, which makes reasoning more difficult [42]. Furthermore, since O occurs only in integrity constraints, facts of the form (60) are sufficient: the minimal model semantics ensures that O contains exactly the individuals $a_1, \ldots, a_n$.

5.3 Data-Related Tasks

After the axioms from the domain model have been correctly separated into the standard TBox $\mathcal{S}$ and the integrity constraint TBox $\mathcal{C}$, we are ready to add data. After appending an ABox $\mathcal{A}$, we then check IC satisfaction using Definition 2. We present algorithms that can be used for this purpose in Section 7.[6] If the ICs are satisfied, then we know that all data satisfies the application's assumptions. Furthermore, by Theorem 1, we can disregard the integrity constraints while computing answers to unions of positive conjunctive queries without the danger of losing some answers.

6 Characterization via Logic Programming

We now develop an alternative characterization of IC satisfaction based on logic programming. Given an extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$, we compute a (possibly disjunctive) stratified logic program that entails a certain atom if and only if $\mathcal{K}$ satisfies $\mathcal{C}$. We first show how to evaluate $\mathcal{C}$ in a model using a stratified datalog program. This result is reminiscent of the Lloyd-Topor transformation of complex formulae in logic programs [28].

Definition 5. *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 in χ. For a first-order formula φ, the* integrity constraint program $\mathsf{IC}(\varphi)$ *is defined recursively as follows, for μ and* sub *as defined in Table 1:*

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

As one can easily see from Definition 5, the program $\mathsf{IC}(\varphi)$ is stratified and nonrecursive. For a finite set of formulae T, we define $\mathsf{IC}(T) = \mathsf{IC}(\varphi)$ where $\varphi = \bigwedge_{\psi \in T} \psi$, and we use E_T as a synonym for E_φ.

Intuitively, the rules in $\mathsf{IC}(\varphi)$ encode the semantics of the propositional connectives and the quantifiers. Thus, when $\mathsf{IC}(\varphi)$ is evaluated in some model I (that contains the fact $HU(a)$ for each element of the domain), the predicate E_χ will contain exactly those facts $E_\chi(\mathbf{t})$ for which $\chi[\mathbf{t}/\mathbf{x}]$ is true in I for each subformula χ of φ. We formalize this as follows.

Lemma 2. *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 $\mathcal{A}(I) \cup \mathsf{IC}(\varphi) \models_c E_\varphi(\mathbf{t})$.*

[6] Integrity constraints can clearly be checked incrementally, while adding facts to the ABox. We consider this an implementation issue and do not consider it any further.

Table 1: The Definition of the Operators μ and sub

	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 $\approx$. *not* is the stratified negation of logic programs.

Proof. The proof is by an easy induction on the structure of φ. For the induction base, if φ is an atomic formula, then $\mathsf{IC}(\varphi)$ contains a rule of the 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 $\mathsf{IC}(\varphi)$ contains a rule of the form (2) from Table 1, which ensures that $E_\varphi(t_1, \ldots, t_n)$ holds exactly if $E_\psi(t_1, \ldots, t_n)$ does not hold. The cases for $\varphi = \psi_1 \wedge \psi_2$ and $\varphi = \psi_1 \vee \psi_2$ are proved in a similar way. For $\varphi = \exists y : \psi$, the program $\mathsf{IC}(\varphi)$ 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 $\mathsf{IC}(\varphi)$ contains a rule of the form (6) which reflects 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. □

Next, we show how to convert the schema $\mathcal{S}$ into an equivalent positive logic program $\mathsf{LP}(\mathcal{S})$.

Definition 6. *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:*

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

Due to the distributive laws for $\wedge$ and $\vee$, $\mathsf{LP}(\varphi)$ can be exponential in the size of φ. Here, we are interested only in the semantic properties of $\mathsf{LP}(\varphi)$; we address the potential exponential blowup in Section 7.1.

We are now ready to present a characterization of IC satisfiability using logic programming.

Theorem 2. *An extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ satisfies the integrity constraints $\mathcal{C}$ if and only if $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{IC}(\mathcal{C}) \models_c E_\mathcal{C}$.*

Proof. For $\varphi = \mathsf{sk}(\mathcal{S})$, the formula φ' in Definition 6 is obtained 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 rules introduced in the third and fourth

item of Definition 6 just enumerate the entire Herbrand universe, so each minimal Herbrand model of $\{\varphi'\} \cup \mathcal{A}$ corresponds exactly to a minimal Herbrand model of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A}$ augmented with $HU(t)$ for each ground term t. The rules of $\mathsf{IC}(\mathcal{C})$ contain only the predicates E_χ in their heads, and each predicate depends only on the predicates corresponding to the subformulae of χ. Hence, the program $\mathsf{IC}(\mathcal{C})$ is stratified, and $\mathsf{IC}(\mathcal{C})$ just extends each minimal model I of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A}$ to a minimal model I' of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{IC}(\psi)$ by facts of the form $E_\chi(\mathsf{t})$, for χ a subformula of $\mathcal{C}$. By Lemma 2, $I' \models E_\mathcal{C}$ if and only if $I' \models \mathcal{C}$, which implies our claim. □

Theorem 2 is significant for two reasons. On the one hand, it provides the foundation for IC satisfaction checking in several practical cases (see Section 7). On the other hand, it provides us with a slightly more procedural intuition about the nature of integrity constraints. Rules of the form $A \rightarrow B$ from $\mathsf{LP}(\mathcal{S})$ do not contain negated atoms, and they can be seen as procedural rules of the form "from A conclude B." Thus, an extended DL knowledge base satisfies the integrity constraints in $\mathcal{C}$ if these are satisfied in each minimal set of facts derivable from $\mathcal{S} \cup \mathcal{A}$.

7 Checking Satisfaction of Integrity Constraints

We now consider algorithms for checking whether an extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ satisfies $\mathcal{C}$. The difficulty of that task is determined primarily by the structure of the standard TBox $\mathcal{S}$. This is because evaluating a formula in a Herbrand model is easy regardless of the formula structure; the difficult task is the computation of the minimal models of $\mathsf{sk}(\mathcal{S} \cup \mathcal{A})$. In the rest of this section, we consider different possibilities for doing so depending on the form of $\mathcal{S}$.

If $\mathcal{S}$ contains no functions symbols, no existential quantifiers under positive polarity, and no universal quantifiers under negative polarity, then we can use Theorem 2: the program $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{IC}(\mathcal{C})$ then does not contain function symbols, so we can use any (disjunctive) datalog engine for checking IC satisfaction. A minor difficulty is caused by the fact that $\mathsf{LP}(\mathcal{S})$ can be exponential in size. Therefore, in Section 7.1, we show how to perform the translation without such a blowup, and we apply this result to existential-free knowledge bases in Section 7.2. Finally, in Section 7.3, we consider schemata expressed in the DL $\mathcal{ALCHI}$.

7.1 Structural Transformation and Minimal Models

It is well-known that the translation into conjunctive normal form, employed in Definition 6, can incur an exponential blowup, which can be avoided by applying the *structural transformation* [34] as follows. For a first-order

formula φ, the result of applying the structural transformation to a single occurrence of a subformula χ is the formula

$$\psi = \begin{cases} \varphi' \wedge \forall x_1, \ldots, x_n : [Q(x_1, \ldots, x_n) \rightarrow \chi] & \text{if } \chi \text{ occurs positively in } \psi \\ \varphi' \wedge \forall x_1, \ldots, x_n : [Q(x_1, \ldots, x_n) \leftarrow \chi] & \text{if } \chi \text{ occurs negatively in } \psi \\ \varphi' \wedge \forall x_1, \ldots, x_n : [Q(x_1, \ldots, x_n) \leftrightarrow \chi] & \text{if } \chi \text{ occurs both positively} \\ & \quad \text{and negatively in } \psi \end{cases}$$

where $x_1, \ldots, x_n$ are the free variables of χ, Q is a fresh predicate, and φ' is obtained from φ by replacing the mentioned occurrence of χ with the atom $Q(x_1, \ldots, x_n)$. With "occurs positively" and "occurs negatively" we mean that χ occurs in ψ under even and odd number of (both explicit and implicit) negations, respectively; furthermore, χ occurs in ψ both positively and negatively if it occurs under the equivalence symbol $\leftrightarrow$. It is well known that this transformation preserves the satisfiability of φ [34].

We illustrate the above definition by means of an example. Consider the formula

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

To transform φ into conjunctive normal form, we would need to distribute $\wedge$ over $\vee$, which would double the size of the formula. To prevent this, we apply the structural transformation to the occurrence of the subformula $\chi = B(y) \vee C(y)$, resulting in

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

If we apply the structural transformation to all nonatomic subformulae of some formula, we can transform the result into conjunctive normal form without an exponential blowup. Furthermore, the transformation is applied at most once to each subformula of a formula, so the result can be computed in polynomial time.

The structural transformation introduces additional symbols, so it is not immediately clear that it preserves the minimal models. Therefore, in the rest of this section we investigate the precise relationship between the minimal models before and after the transformation. We use the following notation. For an interpretation I and a set of predicates Υ, let $I/_\Upsilon$ be the restriction of I to the predicates in Υ, defined as follows:

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

For a formula φ, let $\mathsf{pred}(\varphi)$ be the set of all predicates in φ. We will use $I/_\varphi$ as an abbreviation for $I/_{\mathsf{pred}(\varphi)}$.

Let φ be a formula and ψ a formula obtained from φ through structural transformation. Since this transformation extends the signature of φ, it is clear that φ is not equivalent to ψ. 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 property does not hold. For example, the only minimal Herbrand model of $\varphi_1 = A \wedge C \wedge [A \rightarrow B \vee (C \wedge \neg D)]$ is $I = \{A, C\}$. Applying the structural transformation to φ_1 produces the formula $\psi_1 = A \wedge C \wedge (A \rightarrow B \vee Q) \wedge (Q \rightarrow C \wedge \neg D)$ with the minimal models $I'_1 = \{A, Q, C\}$ 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_1.

To precisely describe the relationship between the formulae before and after the structural transformation, we use the following definition.

Definition 7. *A Herbrand interpretation I is a* Υ-minimal model *of a formula ψ if $I \models \psi$ and $I' \not\models \psi$ for each interpretation I' such that $I'/_\Upsilon \subsetneq I/_\Upsilon$. Furthermore, for a formula φ, the interpretation I is a* φ-minimal model *of ψ if and only if it is a* $\mathsf{pred}(\varphi)$*-minimal model of ψ.*

Thus, an Υ-minimal model is a model in which the extensions of the predicates in Υ are minimal; the extensions of the remaining predicates need not be minimal. As a consequence, the model I'_1 from the previous example is $\mathsf{pred}(\varphi_1)$-minimal, whereas I'_2 is not.

The relationship between the models before and after the structural transformation is described by the following theorem.

Theorem 3. *Let φ be a first-order formula and ψ a formula obtained from φ by applying the structural transformation to an occurrence of a subformula χ. Then,*

1. *for each minimal Herbrand model I of φ, a minimal model I' of ψ exists such that $I = I'/_\varphi$, and*
2. *for each φ-minimal Herbrand model I' of ψ, the interpretation $I'/_\varphi$ is a minimal Herbrand model of φ.*

Proof. Let φ, ψ, and χ be as stated in the theorem. The following properties are well-known [34]: (*) 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 of ψ whose existence is implied by (**). Clearly, a minimal Herbrand model I' of ψ must exist 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 $I \models \varphi$. 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. □

The situation is easier for Horn formulae—disjunctions of literals with at most one positive atom. It is well known that such formulae can have

at most one minimal Herbrand model, so the following corollary follows immediately from Theorem 3.

Corollary 1. *Let ψ be a conjunction of Horn formulae obtained from some formula φ by one or more applications of the structural transformation. If I is a minimal model of ψ, then $I/_{\varphi}$ is a minimal model of φ.*

7.2 Checking IC Satisfaction for Existential-Free KBs

In this section we consider the quite common case of existential-free extended DL knowledge bases, which are defined as follows.

Definition 8. *An extended DL knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ is* existential-free *if no formula in $\pi(\mathcal{S})$ contains a function symbol, an existential quantifier occurring positively, or a universal quantifier occurring negatively.*

Thus, $\mathcal{S}$ can contain an axiom of the form $\forall x : [[\exists y : R(x, y)] \rightarrow C(x)]$, but not an axiom of the form $\forall x : [C(x) \rightarrow \exists y : R(x, y)]$: in the first case, the existential quantifier occurs on the left-hand side of the implication and is effectively equivalent to a universal quantifier, whereas in the second case, the existential quantifier occurs positively in the formula and it implies the existence of individuals in a model. The integrity constraint TBox $\mathcal{C}$ can contain existential quantifiers both under positive and negative polarity; these quantifiers, however, represent requirements on the facts that must be present in the ABox. Hence, all individuals in an existential-free knowledge base are explicitly known by name—that is, it is not necessary to consider unnamed individuals. We expect many data-centric applications of OWL to fall into this category. Existential-free knowledge bases exhibit a useful property that can be exploited in checking IC satisfaction.

Proposition 1. *If $\mathcal{K}$ is existential-free, then $\mathsf{LP}(\mathcal{S})$ does not contain function symbols.*

The restricted way in which function symbols in $\mathsf{LP}(\mathcal{S})$ are introduced makes it is easy to see that this proposition is true—namely, they are introduced only by the Skolemization of the existential quantifiers occurring positively or universal quantifiers occurring negatively in $\mathcal{S}$.

Thus, for existential-free knowledge bases, we can check IC satisfaction using standard logic programming machinery. If the computation of $\mathsf{LP}(\mathcal{S})$ does not incur an exponential blowup, then we do not need the structural transformation, and we can apply Theorem 2 directly. If we apply the structural transformation, but the program $\mathsf{LP}(\mathcal{S})$ is nondisjunctive, we can also use Theorem 2 due to Corollary 1. The problem arises if $\mathsf{LP}(\mathcal{S})$ is disjunctive and we apply the structural transformation: by Theorem 3, we must then consider the Υ-minimal models of $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{IC}(\mathcal{C})$ where Υ is the set of predicates before the structural transformation. Next we show how to check Υ-minimality for propositional formulae.

Theorem 4. *Let Υ be a set of propositional symbols, φ a propositional formula, and I an interpretation such that $I \models \varphi$. Then, I is a Υ-minimal model of φ if and only if $\zeta(\varphi, I, \Upsilon)$, defined as follows, is unsatisfiable:*

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

Proof. For the ($\Rightarrow$) direction, assume that $\zeta(\varphi, I, \Upsilon)$ is satisfiable in a model I'. Due to $\mathsf{neg}(I, \Upsilon)$, if $I \not\models A$ for $A \in \Upsilon$, then $I' \not\models A$ as well; also, 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 $I'/_\Upsilon \subsetneq I/_\Upsilon$, so I is not Υ-minimal. For the ($\Leftarrow$) direction, if I is not a Υ-minimal model of φ, a model I' of φ exists such that $I'/_\Upsilon \subsetneq I/_\Upsilon$. Clearly, I' satisfies both $\mathsf{neg}(I, \Upsilon)$ and $\mathsf{pos}(I, \Upsilon)$, so $\zeta(\varphi, I, \Upsilon)$ is satisfiable. □

Thus, given an existential-free knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$, checking whether $\mathsf{LP}(\mathcal{S}) \cup \mathcal{A} \cup \mathsf{IC}(\mathcal{C}) \models E_\mathcal{C}$ can be performed by grounding the program, guessing a Herbrand interpretation I for it, checking whether I is a model of the program, checking whether I is a Υ-minimal model, and checking whether I does not contain $E_\mathcal{C}$; the minimality check can be performed by checking the satisfiability of $\zeta(\varphi, I, \Upsilon)$. Clearly, the complexity of such an algorithm is not worse than the complexity of disjunctive logic programming: it is in Π_2^p for data complexity and in coNExpTime$^{\text{NP}}$ for combined complexity [17].

We finish this section with a note on equality. Most existing implementations of disjunctive logic programming engines support equality as a built-in predicate that is interpreted as identity and is allowed to occur only in 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. Note that the logic program $\mathsf{IC}(\mathcal{C})$ can also contain equality, but only in rule bodies. Hence, $\mathsf{IC}(\mathcal{C})$ cannot constrain two constants to be equal; it can only check whether two constants have been derived to be equal. If $\mathsf{IC}(\mathcal{C})$ contains equality but $\mathsf{LP}(\mathcal{S})$ does not, then we can simply interpret equality in $\mathsf{IC}(\mathcal{C})$ as identity and use the built-in implementation of equality.

7.3 Checking IC Satisfaction for General KBs

We now consider the problem of checking IC satisfaction when $\mathcal{S}$ contains existentials. This turns out not to be easy, mainly because the Herbrand models of $\mathsf{sk}(\mathcal{S})$ are infinite, so we cannot represent them explicitly.

In this section we present an algorithm for IC satisfaction checking that can be used if $\mathcal{S}$ is expressed in the DL $\mathcal{ALCHI}$. On the one hand, this DL contains constructs characteristic of most DL languages, and on the other hand, the decision procedure does not get too complex. We conjecture that this technique can be extended to handle other constructs found in OWL, such as number restrictions or nominals; however, the technical details would obscure the nature of our result. We do not intend our algorithm to be used in practice; rather, our result should be understood as evidence that checking IC satisfaction is, in principle, possible for nontrivial DLs. Therefore, we leave the development of a more practical procedure as well as extending it to more expressive DLs for future work. Note that we place no restrictions on the form of the ICs in $\mathcal{C}$: they can be arbitrary first-order formulae.

To make this paper self-contained, we start with a formal definition of the DL $\mathcal{ALCHI}$. The basic components of an $\mathcal{ALCHI}$ knowledge base are atomic concepts, which correspond to unary predicates, and atomic roles, which correspond to binary predicates. A *role* is either an atomic role or an *inverse role* R^- for R an atomic role; furthermore, we define $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$. The set of $\mathcal{ALCHI}$ concepts is defined inductively as the smallest set containing the atomic concepts, $\neg C$ (*negation*), $C \sqcap D$ (*conjunction*), $C \sqcup D$ (*disjunction*), $\exists R.C$ (*existential quantification*), and $\forall R.C$ (*universal quantification*), for C and D concepts. An $\mathcal{ALCHI}$ knowledge base $\mathcal{K} = (\mathcal{S}, \mathcal{A})$ consists of a TBox $\mathcal{S}$, which is a set of *general concept inclusion axioms* $C \sqsubseteq D$ for C and D concepts, *role inclusion axioms* $R \sqsubseteq S$ for R and S roles, and an ABox $\mathcal{A}$, which is a set of facts of the form $A(a)$ and $R(a, b)$ for A an atomic concept and R an atomic role. The semantics of $\mathcal{K}$ can be given by translating $\mathcal{S}$ into a first-order formula $\pi(\mathcal{S})$, where the definition of π is given in Table 2.[7]

We embed the problem of checking IC satisfaction into the problem of checking satisfiability of monadic second-order formulae on infinite k-ary trees *SkS* [36]. We use *SkS* because it allows us to encode the tree-like structure of Herbrand models, and it provides for second-order quantification that can be used to express the minimality criterion. *SkS* terms are built as usual from first-order variables (written in lowercase letters), a constant symbol ε, and k unary function symbols f_i. 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 (written in uppercase 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 [36]. Intuitively, first-order quantification ranges over domain elements, whereas second-order quantification ranges over domain subsets. The symbol $=$ denotes true equality in *SkS*, and it is different from

[7] The operators π_x, π_y, π_{xy}, and π_{yx} are mutually recursive, and they reuse the variables x and y for nested expressions.

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

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

the symbol $\approx$ used so far, which denotes a congruence relation on Herbrand models. We use $P \subseteq R$ as an abbreviation for $\forall x : P(x) \rightarrow R(x)$, and $P \subsetneq R$ as an abbreviation for $P \subseteq R \wedge \neg(R \subseteq P)$.

Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base in which $\mathcal{S}$ is an $\mathcal{ALCHI}$ TBox, and let $\psi = \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$. We now show how to compute an *SkS* formula $SkS_{\mathcal{K}}$ that is satisfiable if and only if $\psi \models_{\mathsf{MM}} \mathcal{C}$. The formula ψ contains binary atoms and is therefore not an *SkS* formula. We proceed as follows. First, we observe that ψ contains subformulae of the form shown in Table 3. Next, based on the formula structure, we show that all models of ψ are *forest-like*—that is, they contain binary atoms only of a certain form. Due to their restricted form, we show that such binary atoms can be encoded using unary atoms. Finally, since *SkS* provides only for one constant ε, we encode all constants in ψ using function symbols. The result is an *SkS* formula, and we simply encode the minimality condition using second-order quantifiers.

Without loss of generality, we assume that all concepts in $\mathcal{S}$ are in negation-normal form—that is, that negation occurs only in front of atomic concepts. Then, it is easy to see that the formula $\psi = \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$ can be

Table 3: Skolemization of Concepts in NNF

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 fresh in each invocation of λ.

computed as follows, where λ is the operator from Table 3:

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

We now define different types of models that we consider:

Definition 9. *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. A Herbrand interpretation I is* monadic *if it contains only unary predicates and all function symbols are at most unary.*

One might expect forest-like models to contain the facts of the form $R(a,b)$ instead of facts of the form $R(a,t)$; we discuss the rationale behind our definition after Definition 10. We next prove the core property of the models of ψ.

Lemma 3. *All minimal Herbrand models of the formula $\psi = \mathsf{sk}(\mathcal{S} \cup \mathcal{A})$ are forest-like.*

Proof. If I is a model of ψ 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 9. 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 $\{R \sqsubseteq S, \mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{S}\}$. For the subformula ψ_1 of ψ, it is

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 $f(a)$

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. Note that only positive binary atoms could have a different truth value in I and I'. All such atoms in ψ_2 stem from lines 5 and 6 of Table 3 and are thus of the form $R(t, f(t))$ or $R(f(t), t)$. Therefore, they are included in I' whenever they are included in I. □

For each binary predicate R, function symbol f, and constant a in ψ, we introduce the unary predicates R_f, R_f^-, and R_a in order to encode binary atoms in a forest-like model.

Definition 10. *For a forest-like Herbrand interpretation I, 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 a monadic Herbrand interpretation I, 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 of Definition 10. To be consistent, we might be tempted to encode $R(f(t), t)$ as $R_f^-(f(t))$. Similarly, we might restrict the forest-like interpretations only to atoms of the form $R(a, b)$ instead of $R(a, t)$. But then, we would lose the one-to-one correspondence between forest-like and monadic interpretations: "decoding" a monadic atom $R_f^-(a)$ is not possible because there is no predecessor for a; similarly, "decoding" an atom $R_a(f(f(a)))$ would produce an atom $R(a, f(f(a)))$, which would not be tree-like. Our definitions ensure that each forest-like interpretation can be encoded as a monadic one and vice versa. The requirement in Table 4 that t is not of the form $f(a)$ ensures that the transformation is uniquely defined. We now define an encoding of binary literals, which we then apply in Theorem 5 to the standard TBox and the ICs.

Definition 11. *For R a binary predicate and Σ a set of function symbols and constants, the formula $\nu[R, \Sigma](x, y)$ is defined as follows, where a are constants and f are function symbols:*

$$\nu[R, \Sigma](x, y) = \nu_1[R, \Sigma](x, y) \vee \nu_2[R, \Sigma](x, y) \vee \nu_3[R, \Sigma](x, y)$$

$$\nu_1[R,\Sigma](x,y) = \bigvee_{a\in\Sigma} [x = a \wedge R_a(y)]$$

$$\nu_2[R,\Sigma](x,y) = \bigvee_{f\in\Sigma} [y = f(x) \wedge R_f(x)]$$

$$\nu_3[R,\Sigma](x,y) = \bigvee_{f\in\Sigma} [x = f(y) \wedge R_f^-(y)]$$

For a formula φ, the formula $\nu[\varphi,\Sigma]$ is obtained from φ by replacing each atom $R(s,t)$ with $\nu[R,\Sigma](s,t)$.[8]

Lemma 4. *Let φ be a formula containing only unary and binary predicates, Σ a set containing all constants and function symbols from φ, and $\xi = \nu[\varphi,\Sigma]$. Then,*

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

Proof. For the first claim, we prove a slightly more general property: for φ a formula containing only unary and binary predicates with free variables $\mathbf{x}$, $\xi = \nu[\varphi,\Sigma]$, and $\mathbf{t}$ a vector of terms, we have $I \models \varphi[\mathbf{t}/\mathbf{x}]$ if and only if $\widetilde{I} \models \xi[\mathbf{t}/\mathbf{x}]$. 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 11. Since I is forest-like, $I \models \varphi[\mathbf{t}/\mathbf{x}]$ if and only if $\varphi[\mathbf{t}/\mathbf{x}]$ is of the form $R(a,t)$, $R(t,f(t))$, or $R(f(t),t)$. In the first case, $\widetilde{I}$ satisfies $\nu_1[R,\Sigma](u,v)$; in the second case, it satisfies $\nu_2[R,\Sigma](u,v)$; and in the third case, it satisfies $\nu_3[R,\Sigma](u,v)$. The induction step for Boolean connectives and quantifiers is trivial and is omitted for the sake of brevity. The proof of the second claim is completely equivalent to the proof of the first one. □

Our final obstacle is caused by the fact that *SkS* provides for only one constant ε, while ψ can contain n different constants $a_1,\ldots,a_n$ (w.l.o.g. we assume that $a_i \neq \varepsilon$), so we encode a_i using function symbols.

Definition 12. *Let ψ be a Skolemized formula containing the constants $a_1,\ldots,a_n$ and the function symbols $f_1,\ldots,f_m$, respectively. For $k = m+n$, let $f_{m+1},\ldots,f_k$ be new unary function symbols. For a formula φ, the formula $\mathsf{cs}_\psi(\varphi)$ is 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 an abbreviation for $f_{m+i}(\varepsilon)$. The number k from Definition 12 defines the number of successors of the *SkS* formula we are computing. The following proposition follows trivially from the fact that ε and $f_{m+1},\ldots,f_k$ do not occur in ψ:

[8] We assume that the atoms $R^-(s,t)$ in φ are represented as $R(t,s)$.

Proposition 2. *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 $I' \models \mathsf{cs}_\psi(\varphi)$, for any formula φ.*

To check satisfaction of ICs in $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$, we now construct a formula $SkS_\mathcal{K}$ that is satisfiable if and only if $\mathsf{sk}(\mathcal{S} \cup \mathcal{A}) \models_{\mathsf{MM}} \mathcal{C}$.

Definition 13. *Let $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be an extended DL knowledge base. Then, $SkS_\mathcal{K}$ is the SkS formula defined as follows, where P_i are all the predicates occurring in SkS_α, P'_i are all the predicates occurring in $SkS_{\alpha'}$, and Σ contains all the constants and function symbols occurring in ψ.*

$$
\begin{aligned}
\psi &= \mathsf{sk}(\mathcal{S} \cup \mathcal{A}) \\
\alpha &= \mathsf{cs}_\psi(\psi) \\
&\alpha' \text{ is obtained by replacing each predicate } P \text{ in } \alpha \text{ with a fresh predicate } P' \\
\beta &= \mathsf{cs}_\psi(\mathcal{C}) \\
SkS_\alpha &= \nu[\alpha, \Sigma] \\
SkS_{\alpha'} &= \nu[\alpha', \Sigma] \\
SkS_\beta &= \nu[\beta, \Sigma] \\
SkS_\subsetneq &= (P'_1 \subseteq P_1 \wedge \ldots \wedge P'_n \subseteq P_n) \wedge (P'_1 \subsetneq P_1 \vee \ldots \vee P'_n \subsetneq P_n) \\
SkS_{MM} &= \forall P'_1, \ldots, P'_n : SkS_\subsetneq \rightarrow \neg SkS_{\alpha'} \\
SkS_\mathcal{K} &= \forall P_1, \ldots, P_n : [(SkS_\alpha \wedge SkS_{MM}) \rightarrow SkS_\beta]
\end{aligned}
$$

Intuitively, the outer quantifiers $\forall P_1, \ldots, P_n$ in the formula $SkS_\mathcal{K}$ fix a valuation I of propositional symbols; the formula SkS_α "evaluates" $\mathcal{A} \cup \mathcal{S}$ in I; 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 5. *For $\mathcal{K} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ an extended DL knowledge base, $\psi \models_{\mathsf{MM}} \mathcal{C}$ if and only if $SkS_\mathcal{K}$ 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 4, $\overline{I} \models \alpha$ 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 2, we then have that $\psi \not\models_{\mathsf{MM}} \mathcal{C}$. Assume that $\overline{I}$ is not a minimal model of α—that is, that an interpretation $J \subsetneq \overline{I}$ exists such that $J \models \alpha$. By Lemma 3, J is forest-like. But then, $\widetilde{J} \subsetneq I$, so $\widetilde{J} \models SkS_\subsetneq$; furthermore, by Lemma 4, $\widetilde{J} \models SkS_{\alpha'}$. These two claims now imply that $I \not\models SkS_{MM}$, which is a contradiction.

($\Leftarrow$) If $\psi \not\models_{\mathsf{MM}} \mathcal{C}$, by Proposition 2, we have $\alpha \not\models_{\mathsf{MM}} \beta$. But then, by Lemma 3, a forest-like model I of α exists such that $I \not\models \beta$. By Lemma 4, $\widetilde{I} \models SkS_\alpha$ 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 $J \subsetneq \widetilde{I}$ and $J \models SkS_{\alpha'}$. But then, by Lemma 4, $\overline{J} \models \alpha'$ and $\overline{J} \subsetneq I$, so I is not a minimal model of α. □

Theorem 5 shows that checking IC satisfaction is decidable for nontrivial description logics. Unfortunately, it gives us only a nonelementary upper complexity bound: the complexity of SkS is determined by the number of quantifier alternations [36], which is unlimited because SkS_β can be any first-order formula. In our future work, we shall try to derive tight complexity bounds, as well as a more practical algorithm.

8 Related Work

The usefulness of integrity constraints was recognized early on by the knowledge representation community, and KL-ONE [10] as well as many of its successors provide a variant thereof. For example, CLASSIC [9] provides a kind of derivation rule that can be used to axiomatize ICs. Rules and ICs in such systems, however, typically do not have a formal underpinning. For example, the derivation rules in CLASSIC were given only a procedural semantics that has not been tightly integrated with the core description language.

ICs were given formal semantics in first-order databases using the consistency [24] and the entailment [37] approaches, and these approaches were also applied to closed Prolog-like databases [40, 33, 27]. Reiter conducted a comprehensive study of the nature of ICs [38], in which he argued against all these approaches. He observed that ICs are statements of an epistemic nature; consequently, he proposed to capture their semantics in an extension of first-order logics with an autoepistemic knowledge operator $\mathbf{K}$ that allows an agent to reason about his own knowledge. Lifschitz presented the logic of Minimal Knowledge and Negation-as-Failure (MKNF) which, additionally, provides for a negation-as-failure operator **not** [26].

MKNF was used to obtain an expressive and decidable nonmonotonic DL [16]. One of the motivations for this work was to provide a language capable of expressing integrity constraints. For example, integrity constraint (1) can be expressed using the following axiom (the modal operator $\mathbf{A}$ corresponds to $\neg\,\mathbf{not}$ in MKNF):

$$\mathbf{K}\,Person \sqsubseteq \exists\,\mathbf{A}\,hasSSN\,.\,\mathbf{A}\,SSN \tag{62}$$

MKNF was also used to integrate DLs with logic programming [32]. Again, one of the motivations for this work was to allow for the modeling of integrity constraints. For example, integrity constraint (1) can be expressed by the following logic program:

$$\mathbf{K}\,OK(x) \leftarrow \mathbf{K}\,hasSSN(x,y), \mathbf{K}\,SSN(y) \tag{63}$$

$$\leftarrow \mathbf{K}\,Person(x), \mathbf{not}\,OK(x) \tag{64}$$

Although these existing approaches are motivated similarly to the approach presented in this paper, there are several important differences.

First, rules (63)–(64) do not have any meaning during TBox reasoning; they can only be used to check whether an ABox has the required structure. Axiom (62) can be taken into account during TBox reasoning, but it has a significantly different meaning from our ICs: it only interacts with other modal axioms, but not with the standard first-order axioms. In contrast, the integrity 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 (i.e., as a check). Thus, the semantics of $\mathcal{C}$ is both closer to the standard first-order semantics of description logics, and it mimics more closely the behavior of ICs in relational databases. In our proposal, ICs have a dual role: they describe the domain, as well as the admissible states of the ABox.

Second, the semantics of MKNF makes it difficult to express integrity constraints on unnamed individuals. For a DL 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 explicitly named individuals, and not unnamed individuals implied by existential quantifiers, because in different models one can choose different individuals to satisfy an existential quantifier. Therefore, MKNF-based approaches usually cannot interact with unnamed individuals, so they cannot express the naming constraints mentioned in Section 5.2—that is, they cannot be used to check whether all existentially implied individuals are explicitly named.

Third, MKNF-based integrity constraints work at the level of consequences and are therefore not applicable to disjunctive facts. Consider again the ABox $\mathcal{A}_3$ containing axiom (18) and the standard TBox $\mathcal{S}_3$ containing axiom (19). We might express integrity constraints (20)–(21) using the following MKNF rules:

$$\leftarrow \mathbf{K}\ \mathit{Tiger}(x), \mathbf{not}\ \mathit{Carnivore}(x) \tag{65}$$

$$\leftarrow \mathbf{K}\ \mathit{Leopard}(x), \mathbf{not}\ \mathit{Carnivore}(x) \tag{66}$$

These rules are satisfied in $\mathcal{A}_3 \cup \mathcal{S}_3$ because, roughly speaking, $\mathbf{K}\ \mathit{Tiger}(x)$ can be understood as "$\mathit{Tiger}(x)$ is a consequence." Due to disjunction in axiom (19), neither $\mathit{Tiger}(\mathit{ShereKahn})$ nor $\mathit{Leopard}(\mathit{ShereKahn})$ is a consequence of $\mathcal{A}_3 \cup \mathcal{S}_3$; hence, the premise of neither rule is satisfied and the integrity constraints are not violated.

Integrity constraints are commonly available in object-oriented deductive databases. ConceptBase[9] is such a system, and it is based on the Telos [33] language. IC satisfaction in Telos was formalized similarly as in the consistency approach for closed databases [40]. FLORA-2[10] is an object-oriented deductive database based on F-Logic [23]. F-Logic provides for typing constraints that are formalized in logic programming; furthermore,

[9] `http://dbis.rwth-aachen.de/CBdoc/`

[10] `http://flora.sourceforge.net/florahome.php`

general logic programming rules can be used to express arbitrary checks. OWL-Flight [15] is an ontology langauge that also provides for typing constraints and formalizes them in logic programming. Furthermore, techniques are known that can be used to check satisfaction of such integrity constraints incrementally, after each database update [41].

The encodings of ICs using rules in these approaches are closely related to the one given in Section 6. Just like MKNF-based integrity constraints, however, ICs in object-oriented deductive databases only describe the allowed states of the database and play no role during subsumption reasoning. In fact, DL-style subsumption inferences have generally not been considered in object-oriented databases, so the influence of ICs on such inferences is irrelevant. In contrast, the ability to determine subsumption relationships between schema elements is used in many applications of DLs and OWL, so the role of ICs in subsumption reasoning is much more important.

A further difference is that the languages of object-oriented deductive databases typically do not provide for existential quantification. Languages such as F-Logic allow for function symbols, which can be used to capture our semantics of IC satisfaction based on skolemization (see Section 3). In such a case, however, checking IC satisfaction in these languages easily becomes undecidable due to the presence of arbitrary rules. Decidability of IC satisfaction plays an important role in our work, and in Section 7.3 we identify a case for which decidability is guaranteed.

9 Conclusion

Motivated by the problems encountered in data-centric applications of OWL, we have compared OWL and relational databases w.r.t. their approaches to schema modeling, schema and data reasoning problems, and checking of integrity constraint satisfaction. We have seen that both databases and OWL apply the standard first-order semantics to schema reasoning, and the differences between the two are found mainly in data reasoning problems. In relational databases, answering queries and IC satisfaction checking correspond to model checking, whereas the only form of IC checking available in OWL is checking satisfiability of an ABox w.r.t. a TBox—a problem that is not concerned with the form of the data. This has caused misunderstandings in practice: OWL ontologies can be understood as incomplete databases, while the databases encountered in practice are usually complete.

To control the degree of incompleteness, we have proposed the notion of *extended* DL knowledge bases, in which certain TBox axioms can be designated as integrity constraints. For TBox reasoning, integrity constraints behave just like normal TBox axioms; for ABox reasoning, however, they are interpreted in the spirit of relational databases. We define the semantics

of IC satisfaction such that they indeed check whether all required facts are entailed by the given ABox and TBox.

We have also shown that, if ICs are satisfied, we can disregard them while answering positive queries. This suggests that our semantics of IC satisfaction is indeed reasonable, and it suggests that answering queries with integrity constraints may be easier in practice because one needs to consider only a subset of the TBox. Finally, we have presented an alternative characterization of IC satisfaction based on logic programming, as well as several algorithms for IC satisfaction checking.

The main theoretical challenge for our future research is to derive tight complexity bounds for IC satisfaction checking for knowledge bases with existentials, as well as to define practical algorithms for that case. A more practical challenge is to apply the presented approach in applications and validate its usefulness.

Acknowledgment

We thank the anonymous reviewer for numerous comments that have contributed to the quality of this paper.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] D. Beneventano, S. Bergamaschi, and C. Sartori. Description Logics for Semantic Query Optimization in Object-Oriented Database Systems. *ACM Transactions on Database Systems*, 28(1):1–50, 2003.

[7] 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.

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

[9] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: a structural data model for objects. *ACM SIGMOD Record*, 18(2):58–67, 1989.

[10] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[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, 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.

[13] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[14] K. Clark. Negation as Failure. In H. Gallaire, J. Minker, and J. Nicolas, editors, *Logic and Databases*. Plenum Press, 1978.

[15] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning on the Semantic Web. In *Proc. of the 14th Int. World Wide Web Conference (WWW2005)*, pages 623–632, Chiba, Japan, May 10–14 2005. ACM.

[16] 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.

[17] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[18] 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)*, pages 399–405, India, January 6–12 2007. Morgan Kaufmann Publishers.

[19] 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.

[20] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Conceptual Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):103–137, 2006.

[21] T. Imieliński and W. Lipski. Incomplete Information in Relational Databases. *Jounal of the ACM*, 31(4):761–791, 1984.

[22] M. A. Jeusfeld and M. Staudt. Query Optimization in Deductive Object Bases. In J. C. Freytag, D. Vossen, and G. Maier, editors, *Query Processing for Advanced Database Systems*, pages 145–176. Morgan Kaufmann publishers, August 1993.

[23] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[24] R. A. Kowalski. Logic for Data Description. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103. Plemum Press, 1977.

[25] 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.

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

[27] I. W. Lloyd and R. W. Topor. A Basis for Deductive Database Systems. *Journal of Logic Programming*, 30(1):55–67, 1986.

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

[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, I. Horrocks, and U. Sattler. Bridging the Gap Between OWL and Relational Databases. In *Proc. of the 16th International World Wide Web Conference (WWW 2007)*, pages 807–816, Banff, AB, Canada, May 8–12 2007. ACM Press.

[32] 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)*, pages 477–482, Hyderabad, India, January 6–12 2007. Morgan Kaufmann Publishers.

[33] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *ACM Transactions of Information Systems*, 8(4):325–362, 1990.

[34] 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.

[35] I. Pratt-Hartmann. Complexity of the Guarded Two-variable Fragment with Counting Quantifiers. *Journal of Logic and Computation*, 17(1):133–155, 2007.

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

[37] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In M. L. Brodie, J. Mylopulos, and J. W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, pages 191–233, 1982.

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

[39] 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.

[40] F. Sadri and R. Kowalski. A Theorem-Proving Approach to Database Integrity. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362. Morgan Kaufmann Publishers, 1988.

[41] M. Staudt and M. Jarke. Incremental Maintenance of Externally Materialized Views. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22th Int. Conf. on Very Large Data Bses (VLDB '96)*, pages 75–86, Mumbai, India, September 3–6 1996. Morgan Kaufmann.

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

[43] R. van der Meyden. Logical Approaches to Incomplete Information. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998.