Maintaining Integrity Constraints in Relational to OWL Transformations

Ming Fang, Weiling Li and Rajshekhar Sunderraman
Department of Computer Science
Georgia State University
Atlanta, Georgia 30303
email: {mfang1, wli16}@student.gsu.edu and raj@cs.gsu.edu

Abstract

A large portion of the Semantic Web is expected to be generated from the hidden Web of relational databases. The methods to translate relational content to OWL/RDF are reasonably straightforward, however these methods tend to lose most, if not all, important integrity constraints that are present in the relational data. In this paper, we present a para-consistent logic based method to maintain several important categories of relational integrity constraints while translating relational data into OWL. In addition, we are able to further enforce OWL-like constraints on the resulting RDFs which are essential to fully exploit the expressiveness and logical inference power of OWL. Most importantly, with our method, the problem of inconsistency checking of OWL instances reduces to simple instance matching within two classes.

1. Introduction

As an expressive ontology language, OWL ([16]) has been widely adopted in areas like science and commerce. One reason for its popularity is its ability to formally describe complex concepts and relationships among concepts. More importantly, OWL provides a way to facilitate automated reasoning at both the conceptual and the instance level ([11]). Although numerous ontologies are available in RDF ([8]) form, there is still a huge demand for developing more OWL ontologies for various purposes. With the help of tools such as Protege ([14]), one can always construct an ontology from scratch by first defining concepts and relationships in the TBox, followed by inserting instances of concepts and relationships into the ABox of the ontology. However, when the knowledge base gets huge, this development process may become time-consuming and erroneous.

Alternatively, several approaches that describe mappings of a relational database to a RDF graph are available as well ([6, 5, 7]). The idea of translating relational databases into OWL automatically seems appealing, as relational resources are abundant and of great importance. Taking the contents of the Deep Web into consideration, the market for this idea becomes several orders of magnitude larger ([4]). For one reason, the size of those hidden data is incredibly enormous ([10]), and for the other reason, the owners of these resources might want to take the advantages of the reasoning services in OWL as well. However, current approaches to converting relational information into OWL instances tend to lose most, if not all, vital constraints being enforced in the relational databases.

Consider generic mappings ([9]) as an example. In generic mappings, every tuple \( t \) of a relation with schema \( R \) is represented by a distinct tuple node \( n_t \) in the RDF graph. There will be two types of edges coming out from these tuple nodes. One leads to a class node \( C_R \) (corresponding to the relational schema name of \( R \)) that is used to indicate which class (or table) this node (or tuple) belongs to. The other type of edge is a property edge. For every attribute in \( R \), there will be one edge leading to the value of that attribute in \( t \). This conversion method is problematic because it discards important constraint information such as foreign keys.

Undoubtedly, maintaining integrity constraints in Semantic Web could be as important as in relational databases, especially for database users who wish to exploit automated reasoning in OWL. Integrity constraints are valuable in checking and enforcing data consistency, providing further semantics on data, and promoting semantic query optimization ([15]). However, since OWL employs two features, namely the Open World Assumption (OWA) as well as the Non-Unique Name Assumption (NUNA) ([3]), enforcing constraints in Semantic Web becomes much harder than in relational databases. In OWA, statements cannot be evaluated to be false if they are not explicitly stated in or inferred from the knowledge base. In NUNA, it is possible that two different identifiers refer to the same entity in the knowledge base.

In this paper, we propose a paraconsistent logic based
method to maintain two important categories of constraints, namely the foreign key and the total participation constraints, while translating relational data into OWL. For primary key constraint, minimum cardinality constraint, and maximum cardinality constraint, they can be specified using build-in OWL features. The reader is referred to the conclusion section at the end of this paper. Paraconsistent relational model was proposed in ([2]) in which explicit negative information was stored in a separate (negative) table. The relational algebra was extended to be able to process queries under the Open World Assumption (OWA). Since Semantic Web also uses the OWA as far as negative knowledge is concerned, the paraconsistent data model is suitable to be able to store the ABox knowledge.

Enforcing other OWL-style constraints such as redundant individual type constraints and specific individual type constraints ([18]) is also feasible with our approach. In fact, any integrity constraint of the form

$$A \sqsubseteq C \ (A \text{ is subsumed by } C)$$

is enforceable in our approach, where $A$ is either an atomic class or a compound class built from other classes using connectives such as union, intersection, negation, and etc.. An instance of $A$ will require itself to be present under the class $C$. Many user-defined integrity constraints take this form.

The rest of the paper is organized as follows: We present background information on OWL and paraconsistent relations in Section 2. In Section 3, we formally introduce our paraconsistent logic based method in a step by step manner, along with a example of enforcing foreign keys during relational model was proposed in ([2]) in which explicit negation section at the end of this paper. Paraconsistent reasoning about intensional knowledge as well as ABox reasoning about extensional data. However, reasoning in DL utilizes both OWA and NUNA that make direct data validation difficult. Take the following small knowledge base as an example:

**TBox:**

$$\text{student} \sqsubseteq \text{people}$$

**ABox:**

$$\text{student(a)}, \text{people(a)}$$

The rule in the **TBox** states that concept **student** is subsumed by concept **people**. Attempting to insert **student(b)** without adding **people(b)** into the **ABox** will flag errors in CWA systems such as relational databases. However, in OWA systems like OWL, there will not be a problem because the system will assume the information about **people** is incomplete and will not make negative conclusions based on the lack of **people(b)**. In the cases where OWL is employed to specify integrity constraints while representing a knowledge base, we need somehow close part of the ontology in order to validate the data.

Paraconsistent data model ([2]) could be a good candidate as it operates under OWA but still makes inconsistency checking possible.

**Definition:** Let $\Sigma$ denotes a relation schema that contains a finite set of attributes $A_1, A_2, ..., A_n$ where $n \geq 1$. For each $A_i (1 \leq i \leq n)$, let $D(A_i)$ represents a non-empty domain of values for $A_i$. A tuple $t$ on $\Sigma$ denotes the following mapping $t : \Sigma \rightarrow \bigcup_{A_i \in \Sigma} D(A_i)$ such that for each $A_i (1 \leq i \leq n) \in \Sigma$, $t(A_i) \in D(A_i)$. Let $\tau(\Sigma)$ be the set of all tuples on $\Sigma$. A paraconsistent model on $\Sigma$ is a pair $R= (R^+, R^-)$, where $R^+$ and $R^-$ are any subsets of $\tau(\Sigma)$ and explicitly collect tuples that are known to be in $R$ and known not to be in $R$, respectively.

If a tuple falls beyond the union of $R^+$ and $R^-$, it can only be assumed as unknown. If a tuple appears in both $R^+$ and $R^-$, it indicates an inconsistency. Notice that a paraconsistent structure is strictly general than an ordinary relation. For any ordinary relation, there exists a corresponding paraconsistent relation, but not vice versa. The paraconsistent data model is able to handle incomplete information about tuples, yet still provides a way to check inconsistent data. For the above reasons, paraconsistent model might be suitable to mediate translation from relational data to OWL instances, while keeping the ability to maintain constraints and reject inconsistent data. For details about a complete set of algebraic operators on paraconsistent relations and appropriate handling of storage issues, the reader is referred to ([2]).
3. Maintaining Constraints

Throughout Section 3 and 4, we will demonstrate our ideas using the a small yet efficient relational database of Figure 1. There are four relational tables, namely people, student, course, and take. For simplicity (and without loss of generality) we will assume that each of the "entity" tables (in this case all but the take table) consists of one column containing the primary key value and each "relationship" table (in this case the take table) consists of the primary key columns of the entity types involved in the relationship. Again, for simplicity we will assume that all relationship tables correspond to binary relationships.

For every relationship table $R$ that has two columns (say $A$ and $B$)
Let $C$ be the most general superclass of $A$
(C = $A$ if $A$ does not have any superclass)
Let $D$ be the most general superclass of $B$
($D = B$ if $B$ does not have any superclass)

$$R^- = \{(a, b)|a \in C \text{ and } b \in D \land (a, b) \notin R\}$$

Algorithm 1: Relational to Paraconsistent Translation

Figure 2 shows the negative data tables for the example database.

**Figure 1.** A small relational database with 3 foreign keys: (1) student pointing to people, (2) student column in take pointing to student and (3) course column in take pointing to course

**Figure 2. Negative data tables**

Step 2: Translate positive tables in the relational database into regular classes and properties in OWL. Each tuple of a positive table will become an OWL instance of a class whose name is the same as the corresponding table name in the relational data. For the example database, we will get people(a), people(b), people(c), student(a), student(b), course(s), course(t), take(a,s), and take(b,t).

Step 3: Express integrity constraints as regular Description Logic (DL) axioms. For the first two foreign keys in the sample relational database, the following two DL axioms will be generated: ($\top$ indicates universal concept)

$$\text{student} \sqsubseteq \text{people}$$
$$\exists \text{take}. \top \sqsubseteq \text{student}$$

Step 4: Translate the negative table(s) that corresponds to the righthand side of the DL axioms into new OWL class(es). If the righthand side is in the form of negation (e.g. $\neg C$), still translate the table $C^-$ into a OWL class. For the above example, we will introduce two new OWL classes: people$^-$ and student$^-$ and introduce the instance student$^-$(c).

Step 5 (IC Check): Check whether every non-empty negative class and its positive counterpart are disjoint by simply running the following SPARQL query:

```sparql
ASK{
?x rdf:type C.
?y rdf:type C^-.
FILTER (?x = ?y)
}.
```

3.1 Foreign Key Constraint

Our method to deal with foreign keys is composed of five steps as detailed below:

**Step1:** Convert the given relational database into a paraconsistent database by adding the following tables shown in Figure 2, using Algorithm 1. Missing tuples are considered to be false under the relation predicate due to the Closed World Assumption in relational database, thus these tuples go into negative tables.

For every entity table $R$
Let $S$ be the most general super-class of $R$
(Note: $S = R$ if $R$ does not have any superclass)

$$R^- = \{a|a \in S \text{ and } a \notin R\}$$
End For;

For every relationship table $R$ that has two columns (say $A$ and $B$)
Let $C$ be the most general superclass of $A$
(C = $A$ if $A$ does not have any superclass)
Let $D$ be the most general superclass of $B$
($D = B$ if $B$ does not have any superclass)

$$R^- = \{(a, b)|a \in C \text{ and } b \in D \land (a, b) \notin R\}$$

End For;
If yes, then IC is not violated. In our case, we only need to check if student and student\(^-\) classes in OWL ontology are disjoint. From now on till the end of section 3.1, let us only focus on the second axiom. For example, if we are trying to insert take\((c, t)\) into our ontology, system will attempt to insert student\((c)\) automatically. Then a violation will occur since our ontology contains both student\((c)\) and student\(^-\)(c).

Please note that the reason for not translating negative tables for the lefthand side of the DL axioms (in the case of the second axiom, take\(^-\) table) into OWL classes is that we still want to maintain some flexibility for OWL. Otherwise every insertion of new data into current ontology will cause violation. For example, instead of inserting take\((c, t)\), we insert take\((a, t)\) into OWL, and our OWL is still legal under the second axiom, even though system will attempt to insert student\((a)\) automatically (and will find student\((a)\) is already present). This is consistent with our relational DB because take\((a, t)\) is a valid insertion in relational database as well. However, if take\(^-\) class is also present as the result of translating negative lefthand side tables, insertion of take\((a, t)\) will raise violation because take\(^-\)(a, t) will also be available in our ontology. This false positive indication of violation is obviously unreasonable.

### 3.2 Total Participation Constraint

Our methodology can also easily handle ”Total Participation Constraints” that are present in the relational database. Consider the constraint: Every student must take a course. The first two steps of our methodology to handle such constraints are exactly the same as for foreign key constraints. We now start directly from Step 3 by expressing the integrity constraint as:

\[
\text{student} \subseteq \forall \text{take}.\text{course}
\]

As take is on the righthand side of this axiom, in Step 4, we include a take\(^-\) class and its instances take\(^-\)(a, t), take\(^-\)(b, s), take\(^-\)(c, t), and take\(^-\)(c, s). In Step 5, if we want to insert student\((c)\) into OWL, we will have take\((c, t)\), or take\((c, s)\) or both. Each one of these three cases will raise violation because they all conflict with take\(^-\) class.

### 4. OWL-style Constraints

In addition to maintaining the integrity constraints that were enforced in the original relational database, we are also able to specify and enforce additional OWL-style constraints on the transformed OWL/RDF data. We illustrate two such constraint types with examples.

#### 4.1. Redundant Individual Type Constraint

An OWL-style constraint called redundant individual type constraint is illustrated first. This constraint specifies that an individual cannot be explicitly declared to have both class C and D (where D is a superclass of C) as its types. Suppose \(C \subseteq D\) is already defined in the TBox. Then, in Step 3 this constraint can be expressed as:

\[
C \subseteq \neg D
\]

To enforce this constraint on student class we must check if every instance in student also appears in people\(^-\). If an instance of student does not appear in people\(^-\), a constraint violation should be reported.

#### 4.2. Specific Individual Type Constraint

An OWL-style constraint called Specific Individual Type Constraint requires that the declared type of a given individual in the instance data must be the most specific one. Suppose the following class hierarchy is already defined in the TBox:

\[
C_1 \sqsubseteq C_2 \\
C_2 \sqsubseteq C_3 \\
\ldots \\
C_{i-1} \sqsubseteq C_i \\
\ldots \\
C_{n-1} \sqsubseteq C_n
\]

Then, in Step 3, this constraint can be expressed as follows:

for each \(1 \leq i \leq n - 1\):

\[
C_i \sqsubseteq \neg C_{i+1} \\
C_i \sqsubseteq \neg C_{i+2} \\
\ldots \\
C_i \sqsubseteq \neg C_n
\]

This type of constraint can also be easily checked by membership in positive or negative tables. As an illustration of specific individual type constraint, consider the class hierarchy:

\[
\text{graduateStudent} \sqsubseteq \text{student} \sqsubseteq \text{people}
\]

Let \(\{a, b, c, d\}\) be the set of all individuals and let \(a\) and \(d\) be instances of graduateStudent. This will prohibit \(a\) and \(d\) from being stated as instances of student and people classes. Let \(c\) be an instance of student. This will prohibit \(c\) from being stated as an instance of people class. So, a valid ABox will be:

\[
\text{people}(b) \\
\text{student}(c) \\
\text{graduateStudent}(a) \\
\text{graduateStudent}(d)
\]
If one tries to add student(a), system will detect an inconsistency. This is because a, b, and d will appear in class student due to the IC axiom:

\[
\text{graduateStudent} \subseteq \neg \text{student}
\]

There is an evident conflict between student(a) and student'(a).

5. Related Work

Researchers have constantly shown the importance of mapping from relational databases to OWL in the promotion of the Semantic Web ([5, 17, 6]). Bizer ([5]) introduced a database to RDF mapping language called D2R MAP. In accordance, W3C published new ontology patterns that are used to capture n-ary relations in RDF and OWL, while conventional Semantic Web languages are only capable of representing binary relations ([13]). Although these approaches greatly facilitate the generation of OWL instances, they tend to lose vital constraint information and barely support automated and generic mapping.

One the other hand, efforts have been spent on integrating integrity constraints within RDFs ([18, 12, 11]). The current trend of incorporating ICs into OWL is either based on the query language SPARQL ([18]) or based on logic programming ([11]). Motik et al. ([12]) provide an alternative semantics for integrity constraints in OWL under the minimum Herbrand Model. However, to the best of our knowledge, none of them combines with the relational to OWL mapping and addresses the important relational and OWL aspects of integrity constraints like our approach does. Lausen et al. ([9]) proposed another SPARQL based approach to maintain the primary key and foreign key constraints while translating a relational database to RDF, but their method involves a nonstandard RDF whose vocabulary has been extended.

6. Conclusion and Future Work

We have presented a paraconsistent logic based methodology to translate relational databases into OWL while maintaining important integrity constraints. In general, the method works for foreign key and total participation constraints. It is also possible to write additional OWL-style constraints and have them enforced on the translated OWL data. Thus our method expands the set of integrity constraints that are not embedded in OWL, in addition to the following constraints that can be specified using built-in features of OWL.

Key constraint can be translated using the following two OWL features:

\[
\text{<owl:InverseFunctionalProperty>}
\]
\[
\text{<owl:minCardinality=1>}
\]

The \text{<owl:InverseFunctionalProperty>} tag enforces the unique aspect of primary key constraint, and the \text{<owl:minCardinality=1>} tag enforces the non-empty aspect. Note that, due to the fact the set of object properties and data type properties are disjoint in OWL DL, \text{<owl:InverseFunctionalProperty>} can never be specified for data type properties, otherwise it will exceed the expressive power of OWL DL.

Min cardinality can be directly translated using the following OWL feature:

\[
\text{<owl:minCardinality}
\]
\[
\text{rdf:datatype="&xsd:nonNegativeInteger"}>
\]
\[
\text{n}
\]
\[
</\text{owl:minCardinality}>
\]

Max cardinality can be directly translated using the following OWL feature:

\[
\text{<owl:maxCardinality}
\]
\[
\text{rdf:datatype="&xsd:nonNegativeInteger"}>
\]
\[
\text{n}
\]
\[
</\text{owl:maxCardinality}>
\]

In the future, we would like to extend our method to be able to deal with additional relational-style as well as OWL-style integrity constraints.

References


