On Directly Mapping Relational Databases to RDF and OWL

Juan F. Sequeda¹, Marcelo Arenas², and Daniel P. Miranker¹

¹ Department of Computer Science, The University of Texas at Austin
² Department of Computer Science, Pontificia Universidad Católica de Chile

Abstract. Mapping relational databases to RDF is a fundamental problem for the development of the Semantic Web. We present a solution, inspired by draft methods defined by the W3C where relational databases are directly mapped to RDF and OWL. Given a relational database schema and its integrity constraints, this direct mapping produces an OWL ontology, which, in turn, provides the basis for generating RDF instances. The semantics of this mapping is defined using Datalog. Four fundamental properties of such mappings are monotonicity, information preservation, query preservation and semantics preservation. We prove that our mapping is monotone, information preserving and query preserving. We also prove that no monotone mapping, including ours, is semantics preserving. We realize that monotonicity is an obstacle and thus present a non-monotone direct mapping that is semantics preserving. Additionally, we foresee the existence of monotone direct mappings that are semantics preserving if OWL is extended with the epistemic operator.

1 Introduction

In this paper, we study the problem of directly mapping a relational database to an RDF graph with OWL 2 DL vocabulary. Intuitively, a direct mapping is a default and automatic way to translate a relational database to RDF. One report suggests that Internet accessible databases contain up to 500 times more data compared to the static Web and roughly 70% of websites are backed by relational databases, making automatic translation of relational database to RDF central to the success of the Semantic Web [9].

Several approaches have been presented that directly map relational schemas to OWL and other ontology languages [16]. Currently, the W3C RDB2RDF Working Group is developing a direct mapping standard that focuses on translating relational database instances to RDF [4]. To the best of our knowledge, we are presenting the first direct mapping from a relational database to an RDF graph with OWL 2 DL vocabulary. We build on an existing direct mapping of relational database schema to OWL DL [18]. Additionally, we study four properties that are fundamental to a direct mapping: monotonicity, information preservation, query preservation and semantics preservation.

Monotonicity is a desired property because it assures that a re-computation of the entire mapping is not needed after any inserts to the database. Information preservation speaks to the ability of reconstructing the original database from the result of the direct mapping. Query preservation means that every query over a relational database can be
translated into an equivalent query over the result of the direct mapping. Finally, a direct mapping is semantics preserving if the satisfaction of a set of integrity constraints (i.e. primary keys and foreign keys) are encoded in the mapping result. Our proposed direct mapping is information preserving and query preserving. However, it is not semantics preserving because a database that violates an integrity constraint, when mapped, generates an RDF graph that is consistent.

We analyze why our direct mapping is not semantics preserving and realize that monotonicity is an obstacle. We first show that if we only consider primary keys, we can still have a monotone direct mapping that is semantics preserving. However this result is not sufficient because it dismisses foreign keys. Unfortunately, we prove that no monotone direct mapping is semantics preserving if foreign keys are also considered, essentially because the only form of constraint checking in OWL 2 DL is satisfiability testing. This result has an important implication in real world applications: if you migrate your relational database to the Semantic Web using a monotone direct mapping, be prepared to experience consistency when what one would expect is inconsistency.

Finally, we present a non-monotone direct mapping that overcomes the aforementioned limitation. Additionally, we foresee the existence of monotone direct mappings if OWL 2 DL is extended with the epistemic operator. Due to lack of space, the paper does not include the proofs of the results. We refer the reader to [15] for these proofs.

2 Preliminaries

In this section, we define the basic terminology that is used in the paper.

2.1 Relational databases

Assume given a countably infinite domain $D$. A schema $R$ is a finite set of relation names, where for each $R \in R$, $att(R)$ denotes the nonempty finite set of attributes names associated to $R$. An instance $I$ of $R$ assigns to each relation symbol $R \in R$ a finite set $R^I = \{t_1, \ldots, t_t\}$ of tuples, where each tuple $t_j$ $(1 \leq j \leq t)$ is a function that assigns to each attribute in $att(R)$ a value from $D$. We use notation $t.A$ to refer to the value of a tuple $t$ in an attribute $A$ (instead of the usual notation $t(A)$ for functions).

We consider two types of integrity constraints: keys and foreign keys. Fix a relational schema $R$. A key $\varphi$ over $R$ is an expression of the form $R[A_1, \ldots, A_m]$, where $R \in R$ and $\emptyset \subseteq \{A_1, \ldots, A_m\} \subseteq att(R)$. Given an instance $I$ of $R$, $I$ satisfies key $\varphi$, denoted by $I \models \varphi$, if for every $t_1, t_2 \in R^I$ such that $t_1.A_k = t_2.A_k$ for each $k \in \{1, \ldots, m\}$, it holds that $t_1 = t_2$. A foreign key over $R$ is an expression of the form $R[A_1, \ldots, A_m] \subseteq_{FK} S[B_1, \ldots, B_m]$, where $R, S \in R$, $\emptyset \subseteq \{A_1, \ldots, A_m\} \subseteq att(R)$ and $\emptyset \subseteq \{B_1, \ldots, B_m\} \subseteq att(S)$. Given an instance $I$ of $R$, $I$ satisfies foreign key $\varphi$, denoted by $I \models \varphi$, if $I \models S[B_1, \ldots, B_m]$ and for every tuple $t$ in $R^I$, there exists a tuple $s$ in $S^I$ such that $t.A_k = s.B_k$ for every $k \in \{1, \ldots, m\}$.

Given a relational schema $R$, a set $\Sigma$ of keys and foreign keys is said to be a set of primary keys (PKs) and foreign keys (FKs) over $R$ if: (1) for every $\varphi \in \Sigma$, it holds that $\varphi$ is either a key or a foreign key over $R$, and (2) there are no two distinct keys in $\Sigma$ of the form $R[A_1, \ldots, A_m]$ and $R[B_1, \ldots, B_n]$ (that is, that mention the same relation
name $R$). Moreover, an instance $I$ of $R$ satisfies $\Sigma$, denoted by $I \models \Sigma$, if for every $\varphi \in \Sigma$, it holds that $I \models \varphi$.

In this paper, we extensively use Datalog. We refer the reader to [2] for the syntax and semantics of this language.

### 2.2 RDF and OWL

Assume there are pairwise disjoint infinite sets $I$ (IRIs), $B$ (blank nodes) and $L$ (literals). A tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple, where $s$ is the subject, $p$ is the predicate and $o$ is the object. A finite set of RDF triples is called an RDF graph. Moreover, assume the existence of an infinite set $V$ of variables disjoint from the above sets, and assume that every element in $V$ starts with the symbol `$?`.

In this paper, we consider RDF graphs with OWL 2 DL vocabulary [1], which is a widely used Web ontology language based on description logics. More specifically, we focus here on RDF graphs with OWL 2 DL vocabulary without datatypes. In particular, we say that an RDF graph $G$ is consistent under the OWL 2 DL semantics if a model of $G$ with respect to the OWL 2 DL vocabulary exists (see [1] for a precise definition of the notion of model and the semantics of OWL 2 DL).

### 3 Direct Mappings: Definition and Fundamental Properties

Intuitively, a direct mapping is a default way to translate relational databases into RDF (that is, without any input from the user on how the relational data should be translated). The input of a direct mapping, $M$, is a relational schema $R$, a set $\Sigma$ of PKs and FKs over $R$ and an instance $I$ of $R$. The output is an RDF graph with OWL 2 DL vocabulary.

From now on, assume that $G$ is the set of all possible RDF graphs and $RC$ is the set of all triples $(R, \Sigma, I)$ such that $R$ is a relational schema, $\Sigma$ is a set of PKs and FKs over $R$ and $I$ is an instance of $R$.

**Definition 1 (Direct mapping).** A direct mapping $M$ is a total function from $RC$ to $G$.

In the rest of this section, we introduce four fundamental properties of direct mappings, namely monotonicity, information preservation, query preservation and semantics preservation.

#### 3.1 Monotonicity

Consider two database instances $I_1$ and $I_2$ such that $I_1$ is contained in $I_2$ (denoted by $I_1 \subseteq I_2$). A direct mapping $M$ is considered monotone if for any such pair of instances, the result of mapping $I_2$ contains the result of mapping $I_1$. In other words, if we insert new data to the database, then the elements of the mapping that are already computed are unaltered.

**Definition 2 (Monotone direct mapping).** A direct mapping $M$ is monotone if for every relational schema $R$, set $\Sigma$ of PKs and FKs over $R$, and instances $I_1$, $I_2$ of $R$ such that $I_1 \subseteq I_2$:

$$M(R, \Sigma, I_1) \subseteq M(R, \Sigma, I_2)$$
3.2 Information preservation

Intuitively, a direct mapping is information preserving if it does not lose any information about the relational instance being translated, that is, if there exists a way to recover the original database instance from the RDF graph resulting from the translation process. Formally, assuming that $\mathcal{I}$ is the set of all possible relational instances, we have that:

**Definition 3 (Information preservation).** A direct mapping $\mathcal{M}$ is information preserving if there exist a computable mapping $\mathcal{N} : \mathcal{G} \rightarrow \mathcal{I}$ such that for every relational schema $R$, set $\Sigma$ of PKs and FKs over $R$, and instance $I$ of $R$ satisfying $\Sigma$:

$$N(\mathcal{M}(R, \Sigma, I)) = I$$

Recall that a mapping $\mathcal{N} : \mathcal{G} \rightarrow \mathcal{I}$ is computable if there exists an algorithm that, given $G \in \mathcal{G}$, computes $\mathcal{N}(G)$.

3.3 Query preservation

Intuitively, a direct mapping is query preserving if every query over a relational database can be translated into an equivalent query over the RDF graph resulting from the mapping. That is, query preservation ensures that every relational query can be evaluated using the mapped RDF data.

To formally define query preservation, we focus on relational queries that can be expressed in relational algebra [2], and RDF queries that can be expressed in SPARQL [13, 12]. Due to lack of space, we do not present here the syntax and semantics of SPARQL, and we refer the reader to [13, 12] for their formal definition. We just point out here that the answer of a SPARQL query $P$ over an RDF graph $G$ is a finite set of solution mappings, where a solution mapping $\mu$ is a partial function from the set $V$ of variables to $(I \cup L \cup B)$. Thus, given the mismatch in the formats of the answers of relational algebra and SPARQL queries, we introduce a function $tr$ that allows one to convert tuples into solution mappings. Formally, given a relational schema $R$, a relation name $R \in R$, an instance $I$ of $R$ and a tuple $t \in R^I$, define $tr(t)$ as the solution mapping $\mu$ such that: (1) the domain of $tr$ is $\{?A \mid A \in att(R)\}$, and (2) $\mu(?A) = t.A$ for every $A$ in the domain of $\mu$.

**Example 1.** Assume that a relational schema contains a relation with name STUDENT and attributes SID and NAME. Moreover, assume that $t$ is a tuple in this relation such that $t.SID = 1$ and $t.NAME = John$. Then, $tr(t) = \mu$, where the domain of $\mu$ is $\{?SID, ?NAME\}$, $\mu(?SID) = 1$ and $\mu(?NAME) = John$.

**Definition 4 (Query preservation).** A direct mapping $\mathcal{M}$ is query preserving if for every relational schema $R$, set $\Sigma$ of PKs and FKs over $R$ and relational algebra query $Q$ over $R$, there exists a SPARQL query $Q^*$ such that for every instance $I$ of $R$ satisfying $\Sigma$:

$$tr(Q(I)) = Q^*(\mathcal{M}(R, \Sigma, I))$$
3.4 Semantics preservation

Intuitively, a direct mapping is semantics preserving if the satisfaction of a set of PKs and FKs by a relational database is encoded in the translation process. More precisely, given a relational schema $\mathbf{R}$, a set $\Sigma$ of PKs and FKs over $\mathbf{R}$ and an instance $I$ of $\mathbf{R}$, a semantics preserving mapping should generate from $I$ a consistent RDF graph if $I \models \Sigma$, and it should generate an inconsistent RDF graph otherwise.

**Definition 5 (Semantics preservation).** A direct mapping $\mathcal{M}$ is semantics preserving if for every relation schema $\mathbf{R}$, set $\Sigma$ of PKs and FKs over $\mathbf{R}$ and instance $I$ of $\mathbf{R}$:

$$I \models \Sigma \iff \mathcal{M}(\mathbf{R}, \Sigma, I) \text{ is consistent under the OWL 2 DL semantics.}$$

4 The Direct Mapping $\mathcal{DM}$

In this section, we introduce a direct mapping, $\mathcal{DM}$, that integrates and extends the functionalities of the direct mappings proposed in [18, 4]. $\mathcal{DM}$ is defined as a set of Datalog rules, which are divided in two parts; the first part is used to translate relational schemas, while the second part is used to translate relational instances.

In Section 4.1, we present the predicates that are used to encode the input of $\mathcal{DM}$. In Section 4.2, we represent some intermediate predicates that are used to store an ontology that is generated from the relational schema and the set of PKs and FKs to be translated. In Section 4.3, we present the Datalog rules that generates the OWL 2 DL vocabulary from a relational schema and a set of PKs and FKs. Finally, we present in Section 4.4 the Datalog rules that generates RDF triples from a relational instance.

Through this section, we use the following running example. Consider a relational database for a university. The schema of this database consists of tables $\text{STUDENT(\text{SID}, \text{NAME})}$, $\text{COURSE(\text{CID}, \text{TITLE}, \text{CODE})}$, $\text{DEPT(\text{DID}, \text{NAME})}$ and $\text{ENROLLED(\text{SID}, \text{CID})}$. Each tuple in table $\text{STUDENT}$ contains the number ($\text{SID}$) and name ($\text{NAME}$) of a student. Each tuple in table $\text{COURSE}$ contains the number ($\text{CID}$) and title ($\text{TITLE}$) of a course, and the department ($\text{CODE}$) where this course is given. Each tuple in table $\text{DEPT}$ contains the number ($\text{DID}$) and name ($\text{NAME}$) of a department in the university. Finally, each tuple in table $\text{ENROLLED}$ contains the number of a student ($\text{SID}$) and the number of a course where this student is enrolled ($\text{CID}$). Moreover, we have the following constraints about the schema of the university: $\text{SID}$ is the primary key of $\text{STUDENT}$, $\text{CID}$ is the primary key of $\text{COURSE}$, $\text{DID}$ is the primary key of $\text{DEPT}$, $(\text{SID}, \text{CID})$ is the primary key of $\text{ENROLLED}$, $\text{CODE}$ is a foreign key in $\text{COURSE}$ that references attribute $\text{DID}$ in $\text{DEPT}$, $\text{SID}$ is a foreign key in $\text{ENROLLED}$ that references attribute $\text{SID}$ in $\text{STUDENT}$, and $\text{CID}$ is a foreign key in $\text{ENROLLED}$ that references attribute $\text{CID}$ in $\text{COURSE}$.

4.1 Storing relational databases

Given that the direct mapping $\mathcal{DM}$ is specified by a set of Datalog rules, its input $(\mathbf{R}, \Sigma, I)$ has to be encoded as a set of relations. In this section, we define the predicates that are used to store the triples of the form $(\mathbf{R}, \Sigma, I)$. More precisely, the following predicates are used to store a relational schema $\mathbf{R}$ and a set $\Sigma$ of PKs and FKs over $\mathbf{R}$. 
- **REL(r)**: Indicates that r is a relation name in R; e.g. \textsc{REL("STUDENT")} indicates that STUDENT is a relation name.\(^3\)
- **ATTR(a,r)**: Indicates that a is an attribute in the relation r in R; e.g. \textsc{ATTR("NAME", "STUDENT")} holds in our running example.
- **PK\(_n\)(a\(_1\),...,a\(_n\),r)**: Indicates that \(r[a\(_1\),...,a\(_n\)]\) is a primary key in \(\Sigma\); e.g. \textsc{PK\(_1\)("SID", "STUDENT")} holds in our running example.
- **FK\(_n\)(a\(_1\),...,a\(_n\),r,b\(_1\),...,b\(_n\),s)**: Indicates that \(r[a\(_1\),...,a\(_n\)] \subseteq \text{FK\(_s\)(b\(_1\),...,b\(_n\))}\) is a foreign key in \(\Sigma\); e.g. \textsc{FK\(_1\)("CODE", "COURSE", "DID", "DEPT")} holds in our running example.

Moreover, the following predicate is used to store the tuples in an relational instance I of a relational schema R.

- **VALUE(v,a,t,r)**: Indicates that v is the value of an attribute a in a tuple with identifier t in a relation r (that belongs to R); e.g. a tuple \(t_1\) of table STUDENT such that \(t_1.SID = "1"\) and \(t_1.NAME = "John"\) is stored by using the facts \textsc{VALUE("1", "SID", "id1", "STUDENT")} and \textsc{VALUE("John", "NAME", "id1", "STUDENT")}, assuming that id1 is the identifier of tuple \(t_1\).

### 4.2 Intermediate predicates for storing an ontology

In order to translate a relational database into an RDF graph with OWL 2 DL vocabulary, we first extract an ontology from the relational schema and the set of PKs and FKs given as input. In particular, we classify each relation name in the schema as a class or a binary relation (which is used to represent a many-to-many relationship between entities in an ER/UML diagram), we represent foreign keys as object properties and attributes of relations as data type properties. More specifically, the following predicates are used to store the extracted ontology:

- **CLASS(c)**: Indicates that c is a class.
- **OP\(_n\)(p\(_1\),...,p\(_n\),d,r)**: Indicates that \(p\(_1\),...,p\(_n\) (\(n \geq 1\)) form an object property with domain \(d\) and range \(r\).
- **DTP(p,d)**: Indicates that p is a data type property with domain \(d\).

The above predicates are defined by the following Datalog rules.

**Identifying binary relation.** We define auxiliary predicates that identify binary relations to facilitate identifying classes, object properties and data type properties. Informally, a relation r is a binary relation between two relations s and t if (1) both s and t are different from r, (2) r has exactly two attributes a and b, which form a primary key of r, (3) a is the attribute of a foreign key in r that points to s, (4) b is the attribute of a foreign key in r that points to t, (5) a is not the attribute of two distinct foreign keys in r, (6) b is not the attribute of two distinct foreign keys in r, (7) a and b are not the

\(^3\)As is customary, we use double quotes to delimit strings.
attributes of a composite foreign key in $r$, and (8) relation $r$ does not have incoming foreign keys. In Datalog this becomes:

$$\text{BinRel}(R, S, T) \leftarrow \text{PK}_2(A, B, R), \neg\text{ThreeAttr}(R), \neg\text{TwoFK}(A, R), \neg\text{TwoFK}(B, R), \neg\text{OneFK}(A, B, R), \neg\text{FKto}(R). \quad (1)$$

In a Datalog rule, negation is represented with the symbol $\neg$ and upper case letters are used to denote variables. Thus, the previous rule states that the relation $R$ is a binary relation between two relations $S$ and $T$ if the following conditions are satisfied.

- Expression $\text{PK}_2(A, B, R)$ in (1) indicates that attributes $A$ and $B$ form a primary key of $R$.
- Predicate $\text{ThreeAttr}$ checks whether a relation has at least three attributes, and it is defined as follows from the base predicate $\text{Attr}$:
  $$\text{ThreeAttr}(R) \leftarrow \text{Attr}(X, R), \text{Attr}(Y, R), \text{Attr}(Z, R), X \neq Y, X \neq Z, Y \neq Z.$$  

Thus, expression $\neg\text{ThreeAttr}(R)$ in (1) indicates that $R$ has at least two attributes. Notice that by combining this expression with $\text{PK}_2(A, B, R)$, we conclude that $A$, $B$ are exactly the attributes of $R$.

- Expressions $\text{FK}_1(A, R, C, D)$ and $\text{FK}_1(B, R, D, T)$ in (1) indicate that $A$ is the attribute of a foreign key in $R$ that points to $S$ and $B$ is the attribute of a foreign key in $R$ that points to $T$, respectively.

- Expressions $R \neq S$ and $T \neq T$ in (1) indicate that both $S$ and $T$ are different from relation $R$.

- Predicate $\text{TwoFK}$ checks whether an attribute of a relation is the attribute of two distinct foreign keys in that relation, and it is defined as follows from the base predicate $\text{FK}_1$:
  $$\text{TwoFK}(X, Y) \leftarrow \text{FK}_1(X, Y, U_1, V_1), \neg\text{FK}_1(X, Y, U_2, V_2), U_1 \neq U_2 \neg\text{FK}_1(X, Y, U_1, V_1), \neg\text{FK}_1(X, Y, U_2, V_2), V_1 \neq V_2$$  

Thus, expressions $\neg\text{TwoFK}(A, R)$ and $\neg\text{TwoFK}(B, R)$ in (1) indicate that attribute $A$ is not the attribute of two distinct foreign keys in $R$ and $B$ is not the attribute of two distinct foreign keys in $R$, respectively.

- Predicate $\text{OneFK}$ checks whether a pair of attributes of a relation are the attributes of a composite foreign key in that relation:
  $$\text{OneFK}(X, Y, Z) \leftarrow \text{FK}_2(X, Y, Z, U, V, W) \neg\text{FK}_2(Y, X, Z, U, V, W)$$  

Thus, expression $\neg\text{OneFK}(A, B, R)$ in (1) indicates that attributes $A$, $B$ of $R$ are not the attributes of a composite foreign key in $R$.

- Finally, predicate $\text{FKto}$ checks whether a relation with two attributes has incoming foreign keys:
  $$\text{FKto}(X) \leftarrow \text{FK}_1(U_1, Y, V, X) \neg\text{FK}_2(U_1, U_2, Y, V_1, V_2, X)$$  

Thus, expression $\neg\text{FKto}(R)$ in (1) indicates that relation $R$ does not have incoming foreign keys.
For instance, `BinRel("ENROLLED", "STUDENT", "COURSE")` holds in our running example. Note that there is no condition in the rule (1) that requires \( S \) and \( T \) to be different, allowing binary relations that have their domain equal to their range. Also note that, for simplicity, we assume in the rule (1) that a binary relation \( R \) consists of only two attributes \( A \) and \( B \). However, this rule can be easily extended to deal with binary relations generated from many-to-many relationships between entities in an ER/UML diagram that have more than two attributes.

**Identifying classes.** In our context, a class is any relation that is not a binary relation. That is, predicate `CLASS` is defined by the following Datalog rules:

\[
\text{CLASS}(X) \leftarrow \text{REL}(X), \neg \text{ISBinRel}(X)\\
\text{ISBinRel}(X) \leftarrow \text{BinRel}(X, Y, Z)
\]

In our running example, `CLASS("DEPT")`, `CLASS("STUDENT")` and `CLASS("COURSE")` hold.

**Identifying object properties.** For every \( n \geq 1 \), the following rule is used for identifying object properties that are generated from foreign keys:\(^4\)

\[
\text{OP}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T) \leftarrow \text{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T), \neg \text{ISBinRel}(S)
\]

This rule states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). It should be noticed that this rule excludes the case of binary relations, as there is a special rule for this type of relations (see rule (1)). In our running example, `OP_2("CODE", "DID", "COURSE", "DEPT")` holds as `CODE` is a foreign key in the table `COURSE` that references attribute `DID` in the table `DEPT`.

**Identifying data type properties.** Every attribute in a non-binary relation is mapped to a data type property:

\[
\text{DTP}(A, R) \leftarrow \text{ATTR}(A, R), \neg \text{ISBinRel}(R)
\]

For instance, we have that `DTP("NAME", "STUDENT")` holds in our running example, while `DTP("SID", "ENROLLED")` does not hold as `ENROLLED` is a binary relation.

### 4.3 Translating a relational schema into OWL

In this section, we define the rules that translates a relational database schema into an OWL 2 DL vocabulary. More specifically, we first introduce a series of rules for generating the IRIs and then we present the Datalog rules that generate OWL 2 DL.

---

\(^4\) Notice that although we consider an infinite number of rules in the definition of \(DM\), for every concrete relational database we will need only a finite number of these rules.
Generating IRIs for classes, object properties and data type properties. We introduce a family of rules that produce IRIs for classes, binary relations, object properties and data type properties identified by the mapping (which are stored in the predicates CLASS, BINREL, OP, and DTP, respectively). More precisely, assume given a base IRI base for the relational database to be translated (for example, "http://example.edu/db/"), and assume given a family of built-in predicates CONCAT (n ≥ 2) such that CONCAT(n) has n + 1 arguments and CONCAT(n)(x1, . . . , xn, y) holds if y is the concatenation of the strings x1, . . . , xn. Then by following the approach proposed in [4], DM uses the following Datalog rules to produce IRIs for classes and data type properties:

\[
\text{CLASSIRI}(R, X) \leftarrow \text{CLASS}(R), \text{CONCAT}(\text{base}, R, X) \\
\text{DTPIRI}(A, R, X) \leftarrow \text{DTP}(A, R), \text{CONCAT}(\text{base}, R, \#^*, A, X)
\]

For instance, http://example.edu/db/STUDENT is the IRI for the STUDENT relation in our running example, and http://example.edu/db/STUDENT#NAME is the IRI for attribute NAME in the STUDENT relation (recall that DTP("NAME", "STUDENT") holds in our running example). Moreover, DM uses the following family of Datalog rules to generate IRIs for object properties. First, for object properties generated from binary relations, the following rules is used:

\[
\text{OPIRI}_1(R, S, T, X) \leftarrow \text{BINREL}(R, S, T), \text{CONCAT}(\text{base}, R, X)
\]

Thus, in our running example http://example.edu/db/ENROLLED is the IRI for binary relation ENROLLED. Second, for object properties generated from a foreign key consisting of n attributes (n ≥ 1), the following rule is used:

\[
\text{OPIRI}_2(X_1, . . . , X_n, Y_1, . . . , Y_n, S, T, X) \leftarrow \text{OP}_{2n}(X_1, . . . , X_n, Y_1, . . . , Y_n, S, T), \\
\text{CONCAT}_{4n+4}(\text{base}, S, \text{""}, T, \text{""}, X_1, \text{""}, . . . , X_{n-1}, \text{""}, X_n, \\
\text{""}, Y_1, \text{""}, . . . , Y_{n-1}, \text{""}, Y_n, X)
\]

Thus, given that OP2("CODE", "DID", "COURSE", "DEPT") holds in our running example, IRI http://example.edu/db/COURSE,#CODE,DID is generated to represented the fact that CODE is a foreign key in the table COURSE that references attribute DID in the table DEPT.

Translating relational schemas. The following Datalog rules are used to generate the RDF representation of the OWL vocabulary. First, a rule is used to collect all the classes:

\[
\text{TRIPLE}(U, \text{"rdf:type"}, \text{"owl:Class"}) \leftarrow \text{CLASS}(R), \text{CLASSIRI}(R, U)
\]

Notice that predicate TRIPLE is used to collect all the triples of the RDF graph generated by the direct mapping DM. Second, the following family of rules is used to collect all the object properties (n ≥ 1):

\[
\text{TRIPLE}(U, \text{"rdf:type"}, \text{"owl:ObjectProperty"}) \leftarrow \\
\text{OP}(X_1, . . . , X_n, S, T), \text{OPIRI}_n(X_1, . . . , X_n, S, T, U)
\]
Third, the following rule is used to collect the domains of the object properties \((n \geq 1)\):

\[
\text{TRIPLE}(U, "rdfs:domain", W) \leftarrow \text{OP}_n(X_1, \ldots, X_n, S, T), \\
\text{OP} \text{IRI}_n(X_1, \ldots, X_n, S, T, U), \text{CLASSIRI}(S, W)
\]

Fourth, the following rule is used to collect the ranges of the object properties \((n \geq 1)\):

\[
\text{TRIPLE}(U, "rdfs:range", W) \leftarrow \text{OP}_n(X_1, \ldots, X_n, S, T), \\
\text{OP} \text{IRI}_n(X_1, \ldots, X_n, S, T, U), \text{CLASSIRI}(T, W)
\]

Fifth, the following rule is used to collect all the data type properties:

\[
\text{TRIPLE}(U, "rdf:type", "owl:DatatypeProperty") \leftarrow \\
\text{DTP}(A, R), \text{DTP} \text{IRI}(A, R, U)
\]

Finally, the following rule is used to collect the domains of the data type properties:

\[
\text{TRIPLE}(U, "rdfs:domain", W) \leftarrow \\
\text{DTP}(A, R), \text{DTP} \text{IRI}(A, R, U), \text{CLASSIRI}(R, W)
\]

### 4.4 Translating a database instances into RDF

In this section, we define the rules that map a relational database instance into RDF. More specifically, we first introduce a series of rules for generating the IRIs and then present the Datalog rules that generate RDF.

**Generating IRIs for tuples.** As mentioned before, our mappings needs to provide a way to generate IRIs for the generated RDF triples. We introduce a family of predicates that produce IRIs for the tuples being translated, where we assume a given a base IRI `base` for the relational database (for example, "http://example.edu/db/").

First, \(\text{DM}\) uses the following Datalog rule to produce IRIs for the tuples of the relations having a primary key:

\[
\text{ROI} \text{RI}_n(V_1, V_2, \ldots, V_n, A_1, A_2, \ldots, A_n, T, R, X) \leftarrow \text{PK}_n(A_1, A_2, \ldots, A_n, R), \\
\text{VALUE}(V_1, A_1, T, R), \text{VALUE}(V_2, A_2, T, R), \ldots, \text{VALUE}(V_n, A_n, T, R), \\
\text{CONCAT}_{n+2}(\text{base}, R, ",", A_1, ",=", V_1, ",", A_2, ",=", V_2, ",", \ldots, ",", A_n, ",=", V_n, X)
\]

Thus, given that the facts \(\text{PK}_1(\"SID\", \"STUDENT\")\) and \(\text{VALUE}(\"1\", \"SID\", \"id1\", \"STUDENT\")\) hold in our running example, we have that the IRI http://example.edu/db/STUDENT#SID=1 is the identifier for the tuple in table \(\text{STUDENT}\) with value 1 in the primary key. Moreover, \(\text{DM}\) uses the following rule to generate blank nodes for the tuples of the relations not having a primary key:

\[
\text{BLANKNODE}(T, R, X) \leftarrow \text{VALUE}(V, A, T, R), \text{CONCAT}(\"-\", R, T, X)
\]
Translating relational instances. The direct mapping $\mathcal{DM}$ generates three types of triples when translating a relational instance: Table triples, reference triples and literal triples [4]. Next we present the Datalog rules for each one of these cases.

In the first place, $\mathcal{DM}$ produces for each tuple $t$ in a relation $R$, a triple indicating that $t$ is of type $R$, which is called a table triple. To construct these tuples, $\mathcal{DM}$ uses the following auxiliary rules:

$$\text{TUPLEID}(T, R, X) \leftarrow \text{CLASS}(R), \text{PK}_n(A_1, \ldots, A_n, R),$$
$$\text{VALUE}(V_1, A_1, T, R), \ldots, \text{VALUE}(V_n, A_n, T, R),$$
$$\text{ROWIRI}_n(V_1, \ldots, V_n, A_1, \ldots, A_n, T, R, X)$$

That is, $\text{TUPLEID}(T, R, X)$ generates the identifier $X$ of a tuple $T$ of a relation $R$, which is an IRI if $R$ has a primary key or a blank node otherwise. Notice that in the preceding rules, predicate $\text{HASPK}_n$ is used to check whether a table $R$ with $n$ attributes has a primary key (thus, $\neg \text{HASPK}_n(R)$ indicates that $R$ does not have a primary key). Predicate $\text{HASPK}_n$ is defined by the following $n$ rules:

$$\text{HASPK}_n(R) \leftarrow \text{PK}_1(A_1, X)$$
$$\ldots$$
$$\text{HASPK}_n(R) \leftarrow \text{PK}_n(A_1, \ldots, A_n, X)$$

Then we have that the following rule generates the table triples:

$$\text{TRIPLE}(U, \text{"rdf:type"}, W) \leftarrow \text{VALUE}(V, A, T, R), \text{TUPLEID}(T, R, U), \text{CLASSIRI}(R, W)$$

For example, the following is a table triple in our running example:

$$\text{TRIPLE("http://example.edu/db/STUDENT#SID=1","rdf:type","http://example.edu/db/STUDENT")}$$

In the second place, $\mathcal{DM}$ generates triples that store the references generated by binary relations and foreign keys, which are called reference triples. More precisely, the following Datalog rule is used to construct reference triples for object properties that are generated from binary relations:

$$\text{TRIPLE}(U, V, W) \leftarrow \text{BINREL}(R, S, T), \text{PK}_2(A, B, R),$$
$$\text{FK}_1(A, R, C, S), \text{VALUE}(V_1, A, T_1, R), \text{VALUE}(V_1, C, T_2, S),$$
$$\text{FK}_1(B, R, D, T), \text{VALUE}(V_2, B, T_1, R), \text{VALUE}(V_2, D, T_3, T),$$
$$\text{TUPLEID}(T_2, S, U), \text{OPIRI}_1(R, S, T, V), \text{TUPLEID}(T_3, T, W)$$

Moreover, the following Datalog rule is used to construct reference triples for object properties that are generated from foreign keys ($n \geq 1$):

$$\text{TRIPLE}(U, V, W) \leftarrow \text{OP}_2n(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T),$$
$$\text{VALUE}(V_1, A_1, T_1, S), \ldots, \text{VALUE}(V_n, A_n, T_1, S),$$
$$\text{VALUE}(V_1, B_1, T_2, T), \ldots, \text{VALUE}(V_n, B_n, T_2, T),$$
$$\text{TUPLEID}(T_1, S, U), \text{OPIRI}_2n(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T, V),$$
$$\text{TUPLEID}(T_2, T, W)$$
Finally, $\mathcal{DM}$ produces for every tuple $t$ in a relation $r$ and for every attribute $a$ of $r$, a triple storing the value of $t$ in $a$, which is called a literal triple. More precisely, the following Datalog rule is used to generate such triples:

\[
\text{TRIPLE}(U, V, W) \leftarrow \text{DTP}(A, R), \text{VALUE}(W, A, T, R), \text{TUPLEID}(T, R, U), \text{DTPIRI}(A, R, V)
\]

The following is an example of a literal triple:

\[
\text{TRIPLE}("\text{http://example.edu/db/STUDENT#SID=1"},
    "\text{http://example.edu/db/STUDENT#NAME"}, "John")
\]

5 Fundamental Properties of the Direct Mapping $\mathcal{DM}$

In this section, we study our direct mapping $\mathcal{DM}$ with respect to the fundamental properties of monotonicity, information preservation, query preservation and semantics preservation defined in Section 3.

First, it is straightforward to see that $\mathcal{DM}$ is monotone, because all the negative atoms in the Datalog rules defining $\mathcal{DM}$ refer to the schema, the PKs and the FKs of the database, and these elements are kept fixed when checking monotonicity. Second, we show that $\mathcal{DM}$ does not lose any piece of information in the relational instance being translated:

\textbf{Theorem 1.} The direct mapping $\mathcal{DM}$ is information preserving.

The proof of this theorem is straightforward, and it involves providing a computable mapping $N : G \rightarrow I$ that satisfies the condition in Definition 3, that is, a computable mapping $N$ that can reconstruct the initial relational instance from the generated RDF graph.

Third, we show that the way $\mathcal{DM}$ maps relational data into RDF allows one to answer a query over a relational instance by translating it into an equivalent query over the generated RDF graph.

\textbf{Theorem 2.} The direct mapping $\mathcal{DM}$ is query preserving.

The proof of this theorem builds on the results of [3], where it is shown that non-recursive Datalog with safe negation (which is as expressive as relational algebra) has the same expressive power as SPARQL.

Finally, the following example shows that the direct mapping $\mathcal{DM}$ is not semantics preserving.

\textbf{Example 2.} Assume that a relational schema contains a relation with name \texttt{STUDENT} and attributes \texttt{SID}, \texttt{NAME}, and assume that the attribute \texttt{SID} is the primary key. Moreover, assume that this relation has two tuples, $t_1$ and $t_2$ such that $t_1$.\texttt{SID} = 1, $t_1$.\texttt{NAME} = John and $t_2$.\texttt{SID} = 1, $t_2$.\texttt{NAME} = Peter. It is clear that the primary key is violated, therefore the database is inconsistent. However, it is not difficult to see that after applying $\mathcal{DM}$, the resulting RDF graph is consistent. $\square$
In fact, the result in Example 2 can be generalized as it is possible to show that the direct mapping $DM$ always generates a consistent RDF graph, hence, it cannot be semantics preserving.

**Proposition 1.** The direct mapping $DM$ is not semantics preserving.

Does this mean that our direct mapping is incorrect? What could we do to create a direct mapping that is semantics preserving? These problems are studied in depth in the following section.

### 6 Semantics Preservation of Direct Mappings

In this section, we study the problem of generating a direct mapping that is semantics preserving. Specifically, we show in Section 6.1 that a simple extension of the direct mapping $DM$ can deal with primary keys. Then we show in Section 6.2 that dealing with foreign keys is more difficult, as any direct mapping that satisfies the natural condition of being monotone cannot be semantics preserving. Finally, we present in Section 6.2 two possible ways of overcoming this limitation.

#### 6.1 A semantics preserving direct mapping for primary keys

Consider a new direct mapping $DM_{pk}$ that extends $DM$ as follows. A Datalog rule is used to determine if the value of a primary key attribute is repeated. If such a violation is found, then an artificial triple is generated that would produce an inconsistency. For example, the following rule is used to map a single attribute primary key:

\[
\text{TUPLE}(a, "\text{owl:differentFrom}"), a) \leftarrow \text{PK}_1(X, R), \text{VALUE}(V, X, T_1, R), \text{VALUE}(V, X, T_2, R), T_1 \neq T_2
\]

In the previous rule, $a$ is any valid IRI. If we apply $DM_{pk}$ to the database of Example 2, it is straightforward to see that starting from an inconsistent relational database, one obtains an RDF graph that is also inconsistent. In fact, we have that:

**Proposition 2.** The direct mapping $DM_{pk}$ is monotone, and it is semantics preserving if one considers only PKs. That is, for every relational schema $R$, set $\Sigma$ of (only) PKs over $R$ and instance $I$ of $R$:

\[
I \models \Sigma \iff DM_{pk}(R, \Sigma, I) \text{ is consistent under the OWL 2 DL semantics.}
\]

A natural question at this point is whether $DM_{pk}$ can also deal with foreign keys. Unfortunately, it is easy to construct an example that shows that this is not the case. Does this mean that we cannot have a direct mapping that is semantics preserving and considers foreign keys? We show in the following section that monotonicity has been one of the obstacles to obtain such a mapping.
6.2 Semantics preserving direct mappings for primary keys and foreign keys

The following theorem shows that the desirable condition of being monotone is, unfortunately, an obstacle to obtain a semantics preserving direct mapping.

**Theorem 3.** No monotone direct mapping \( M \) is semantics preserving.

It is important to understand the reasons why we have not been able to create a semantics preserving direct mapping. The issue is with OWL because of two characteristics: (1) it adopts the Open World Assumption (OWA), where a statement cannot be inferred to be false on the basis of failing to prove it, and (2) it does not adopt Unique Name Assumption (UNA), where two different names can identify the same thing. On the other hand, a relational database adopts the Closed World Assumption (CWA), where a statement is inferred to be false if it is not known to be true, which is the opposite of OWA. In other words, what causes an inconsistency in a relational database, can cause an inference of new knowledge in OWL.

In order to preserve the semantics of the relational database, we need to make sure that whatever causes an inconsistency in a relational database, is going to cause an inconsistency in OWL. Following this idea, we now present a non-monotone direct mapping, \( DM_{pk+fk} \), which extends from \( DM_{pk} \), and checks if there is a violation of the FK integrity constraint beforehand. If such a FK violation exists, then it creates a artificial RDF triple which will generate an inconsistency with respect to OWL 2 DL semantics. More precisely, the following family of Datalog rules are used in \( DM_{pk+fk} \) to detect an inconsistency in a relational database:

\[
\text{VIOLATION}(S) \leftarrow \text{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T), \\
\text{VALUE}(V_1, X_1, T, S), \ldots, \text{VALUE}(V_n, X_n, T, S), \\
\neg \text{ISVALUE}_n(V_1, \ldots, V_n, Y_1, \ldots, Y_n, T)
\]

In the preceding rule, the predicate \( \text{ISVALUE}_n \) is used to check whether a tuple in a relation has values for some given attributes. The predicate \( \text{ISVALUE}_n \) is defined by the following rule:

\[
\text{ISVALUE}_n(V_1, \ldots, V_n, B_1, \ldots, B_n, S) \leftarrow \text{VALUE}(V_1, B_1, T, S), \ldots, \text{VALUE}(V_n, B_n, T, S)
\]

Finally, the following family of Datalog rules is used to obtain an inconsistency in the generated RDF graph:

\[
\text{TRIPLE}(a, "\text{owl:differentFrom"}, a) \leftarrow \text{VIOLATION}(S)
\]

In the previous rule, \( a \) is any valid IRI. It should be noticed that \( DM_{pk+fk} \) is non-monotone because if new data in the database is added which now satisfies the FK constraint, then the artificial RDF triple needs to be retracted.

**Theorem 4.** The direct mapping \( DM_{pk+fk} \) is semantics preserving.

Nevertheless, if we want a semantics preserving monotone direct mapping, we would need to consider an alternative semantics of OWL for expressing integrity constraints. Because OWL is based on Description Logic, we would need a version of DL that supports integrity constraints, which is not a new idea. Integrity constraints are epistemic
in nature and are about “what the knowledge base knows” [14]. Extending DL with the epistemic operator $K$ has been studied [5–7]. Grimm et al. proposed to extend the semantics of OWL to support the epistemic operator [8]. Motik et al. proposed to write integrity constraints as standard OWL axioms but interpreted with different semantics for data validation purposes [11]. Tao et al. showed that integrity constraint validation can be reduced to SPARQL query answering [17]. Recently, Mehdi et al. introduced a way to answer epistemic queries to restricted OWL 2 DL ontologies [10]. Therefore, it is possible to extend $DM_{pk}$ to create a monotone direct mapping that is semantics preserving, but it is based on a non-standard version of OWL including the epistemic operator $K$.

7 Conclusions

In this paper, we study how to directly map relational databases to an RDF graph with OWL 2 DL vocabulary based on four fundamental properties: monotonicity, information preservation, query preservation and semantics preservation. We first present a monotone, information preserving and query preserving direct mapping. Then we prove that the combination of monotonicity with the OWL 2 DL semantics is an obstacle to generate a semantics preserving direct mapping. Finally, we overcome this obstacle by presenting a non-monotone direct mapping that is semantics preserving, and also by discussing the possibility of generating a monotone mapping that assumes an extension of OWL 2 DL with the epistemic operator.

It is important to mention that, for the sake of readability, we did not consider datatypes. However, extending our direct mapping to consider datatypes is straightforward. Additionally, we consider only relational databases with set semantics. However, notice that in our setting each tuple has its own identifier, which is represented in the $VALUE$ predicate. Thus, even if repeated tuples exist, each tuple will still have its unique identifier and, therefore, exactly the same rules can be used to map relational data under bag semantics.

References

A Additional Terminology

A.1 SPARQL

We use SPARQL as the query language for RDF data. The official syntax of SPARQL considers operators `OPTIONAL`, `UNION`, `FILTER`, `SELECT` and concatenation via a point symbol (.), to construct graph pattern expressions [13]. In order to avoid ambiguities in the parsing, we follow the approach proposed in [12], and we present the syntax of SPARQL queries in a more traditional algebraic formalism using operators `AND`, `UNION`, `OPT` (OPTIONAL), `FILTER` and `SELECT`. More precisely, a SPARQL graph pattern expression is defined recursively as follows: (1) a tuple from \((I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)\) is a graph pattern (a triple pattern), (2) if \(P_1\) and \(P_2\) are graph patterns, then expressions \((P_1 \text{ AND } P_2)\), \((P_1 \text{ OPT } P_2)\), and \((P_1 \text{ UNION } P_2)\) are graph patterns, (3) if \(P\) is a graph pattern and \(R\) is a SPARQL built-in condition, then the expression \((P \text{ FILTER } R)\) is a graph pattern, where a SPARQL built-in condition is a Boolean combination of the equality symbol (=) and unary built-in predicate `bound` (see [13] for a complete list of built-in predicates in SPARQL). Moreover, if \(P\) is a graph pattern and \(W\) is a set of variables contained in the set of variables mentioned in \(P\), then \((\text{SELECT } W P)\) is a SPARQL query.

We refer the reader to [12] for the formal definition of the semantics of SPARQL. We just point out here that the answer of a SPARQL query \(P\) over an RDF graph \(G\) is a finite set of solution mappings, where a solution mapping \(\mu\) is a partial function from \(V\) to \((I \cup L \cup B)\).

B Proofs and Intermediate Results

B.1 Proof of Theorem 1

We show that \(DM\) is information preserving by providing a computable mapping \(N: G \rightarrow I\) that satisfies the condition in Definition 3. More precisely, given a relational schema \(R\), a set \(\Sigma\) of PKs and FKs and an instance \(I\) of \(R\) satisfying \(\Sigma\), next we should how \(N(G)\) is defined for \(M(R, \Sigma, I) = G\).

- **Step 1:** Identify all the ontological class triples (i.e `TRIPLE(r_i, "rdf:type", "owl:Class")`). The IRI \(r_i\) identifies an ontological class \(R'_i\). For every \(R'_i\) that was retrieved from \(G\), map it to a relation name \(R_i\).

- **Step 2:** Identify all the datatype triples of a given class (i.e `TRIPLE(d_j, "rdf:type", "owl:DatatypeProperty"), TRIPLE(d_j, "rdfs:domain", r_i)`). The IRI \(d_j\) identifies the datatype property \(D'_j\) and the IRI \(r_i\) identifies the ontological class \(R'_i\) that is the domain of \(D'_j\). Every datatype property \(D'_j\) with domain \(R'_i\) is mapped to an attribute a \(D_j\) of relation name \(R_i\).

- **Step 3:** For each class \(R\) and the datatype properties \(D_1', \ldots, D_n'\) that have domain \(R\), we can recover the instances of relation \(R\) with the following SPARQL query:

\[
Q_1 = (\text{SELECT } {?D_1, \ldots, ?D_n} \ ((?x, "\text{rdf:}type", r_i) \text{ AND } ((?x, d_1, ?D_1) \text{ OPT } \cdots \text{ OPT } (?x, d_n, ?D_n))))
\]
Step 4: Identify all the object property triples (i.e. TRIPLE(s, "rdf:type", "owl:ObjectProperty"). The IRI s that is the result of OP_IRI, identifies the object property $S'$ in the ontology that was originally mapped from a binary relation. This object property $S'$ is mapped back to a binary relation name $S$. Given the definition of a binary relation, we know that $S$ only has two attributes $P$ and $Q$ where $P$ is a foreign key referencing an attribute $X$ of a relation $U$ and $Q$ is a foreign key referencing an attribute $Y$ of a relation $V$. From the triples TRIPLE(s, "rdfs:domain", u) and TRIPLE(s, "rdfs:range", v), the IRI $u$ identifies the ontological class $U'$ which is mapped to the relation $U$ and the IRI $v$ identifies the ontological class $V'$ which is mapped to the relation $V$. The attribute $X$ of relation $U$ is mapped to a datatype property $X'$ with domain $U'$ and IRI $x$. The attribute $Y$ of relation $V$ is mapped to a datatype property $Y'$ with domain $V'$ and IRI $y$. We can now recover the instances of the relation $S$ with the following SPARQL query:

$$Q_2 = (SELECT \{?P, ?Q\} ((?a, s, ?b) AND (?a, s, ?P) AND (?b, y, ?Q)))$$

Step 5: Given that the result of a SPARQL query is a set $\Omega$ of solution mapping $\mu$, we need to translate each solution mapping $\mu \in \Omega$ into a tuple $t$. We define a function $tr^{-1}$ as the inverse of function $tr$, that is, for each solution mapping $\mu$ and variable $?A$ in the domain of $\mu$, $tr^{-1}$ assigns the value of $\mu(?A)$ to $t.A$. Then the mapping function $N$ over $G$ is defined as the following relational instance. For every relation name identified in Steps 1, 2, 3, define $R^N(G)$ as $tr^{-1}(Q_1(G))$, and for every binary relation $S$ identified in Step 4, define $S^N(G)$ as $tr^{-1}(Q_2(G))$.

It is straightforward to prove that for every relational schema $R$, set $\Sigma$ of PKs and FKS and an instance $I$ of $R$ satisfying $\Sigma$, it holds that $N(M(R, \Sigma, I)) = I$. This concludes the proof of the theorem.

B.2 Proof of Theorem 2

To show that DM is query preserving, we have to show that for every relational schema $R$, set $\Sigma$ of PKs and FKS over $R$ and relational algebra query $Q$, there exists a SPARQL query $Q^*$ such that for every instance $I$ of $R$ satisfying $\Sigma$:

$$tr(Q(I)) = Q^*(M(R, \Sigma, I))$$

(2)

The proof builds on the results of Angles and Gutierrez [3], where they showed that non-recursive Datalog with negation is contained in SPARQL. First, given any relational algebra query $Q$, translate it to a Datalog program $H$. Angles and Gutierrez introduce an algorithm to transform Datalog rules into SPARQL graph patterns. This algorithm translates a predicate $p$ in a relational schema $R$ into a graph pattern which returns all tuples in $p$ (Step 6 of Algorithm 3 in [3]). For our proof, we need to replace that step with a new condition and two different graph patterns: if the extensional predicate $p$ is a non-binary relation, then a graph pattern $gp_1$ is used, else if the extensional predicate is a binary relation, then a graph pattern $gp_2$ is used. Graph patterns $gp_1$ and $gp_2$ are defined as follows.
– **Graph Pattern** \(gp_1\) **for Non-binary Relations**: Given an extensional predicate \(r\), which comes from a non-binary relation \(R\) such that \(\text{att}(R) = \{A_1, \ldots, A_n\}\), the direct mapping \(DM\) maps \(R\) to an ontological class \(R'\) with IRI \(r'\) and each attribute \(A_i\) is mapped to a datatype property \(A'_i\) with IRI \(a_i\) and domain \(R'\). Therefore, the graph pattern \(gp_1\) that returns all tuples in \(r\) is the following:

\[
gp_1 = (\text{SELECT} \{?A_1, \ldots, ?A_n\} ((?x, \text{rdf:type}, r') \text{ AND } (?x, a_1, ?A_1) \text{ AND } \cdots \text{ AND } (?x, a_n, ?A_n)))
\]

– **Graph Pattern** \(gp_2\) **for Binary Relations**: Given an extensional predicate \(s\), which comes from a binary relation \(S\) which has only two attributes \(P\) and \(Q\), where \(P\) is a foreign key referencing an attribute \(X\) of a relation \(U\) and \(Q\) is a foreign key referencing an attribute \(Y\) of a relation \(V\), the direct mapping \(DM\) maps the relation \(S\) to an object property \(S'\) with IRI \(s'\), the relation \(U\) and \(V\) to an ontological class \(U'\) and \(V'\) respectively. Furthermore, the attribute \(X\) is mapped to a datatype property \(X'\) with domain \(U'\) and IRI \(x\) and the attribute \(Y\) is mapped to a datatype property \(Y'\) with domain \(V'\) and IRI \(y\). Therefore, the graph pattern \(gp_2\) that returns all tuples in \(s\) is the following:

\[
\text{gp}_2 = (\text{SELECT} \{?P, ?Q\} ((?a, s', ?b) \text{ AND } (?a, x, ?P) \text{ AND } (?b, y, ?Q)))
\]

The rest of algorithm in [3] is not changed. It is straightforward to prove that (2) holds given the definitions of \(gp_1\) and \(gp_2\), which concludes the proof of the theorem.

**B.3 Proof of Proposition 1**

Assume that we have a relational schema containing a relation with name STUDENT and attributes SID, NAME, and assume that the attribute SID is the primary key. Moreover, assume that this relation has two tuples, \(t_1\) and \(t_2\) such that \(t_1.SID = 1, t_1.NAME = John\) and \(t_2.SID = 1, t_2.NAME = Peter\). It is clear that the primary key is violated, therefore the database is inconsistent. If \(DM\) would be semantics preserving, then the resulting RDF graph would be inconsistent under OWL 2 DL semantics. However, the result of applying \(DM\), returns the following consistent RDF graph (assuming given a base IRI *http://example.edu/db/* for the mapping):
TRIPLE("http://example.edu/db/STUDENT",
    "rdf:type", "owl:Class")
TRIPLE("http://example.edu/db/STUDENT#NAME",
    "rdf:type", "owl:DatatypeProperty")
TRIPLE("http://example.edu/db/STUDENT#NAME",
    "rdfs:domain", "http://example.edu/db/STUDENT")
TRIPLE("http://example.edu/db/STUDENT#SID",
    "rdf:type", "owl:DatatypeProperty")
TRIPLE("http://example.edu/db/STUDENT#SID",
    "rdfs:domain", "http://example.edu/db/STUDENT")
TRIPLE("http://example.edu/db/STUDENT#SID=1",
    "http://example.edu/db/STUDENT#NAME",
    "John")
TRIPLE("http://example.edu/db/STUDENT#SID=1",
    "http://example.edu/db/STUDENT#NAME",
    "Peter")
TRIPLE("http://example.edu/db/STUDENT#SID=1",
    "http://example.edu/db/STUDENT#SID",
    "1")

Therefore, $\mathcal{DM}$ is not semantics preserving. □

B.4 Proof of Proposition 2

It is straightforward to see that given a relational schema $R$, set $\Sigma$ of (only) PKs over $R$ and instance $I$ of $R$ such that $I \models \Sigma$, it holds that $\mathcal{DM}_{pk}(R, \Sigma, I)$ is consistent under the OWL 2 DL semantics. Likewise, if $I \not\models \Sigma$, then by definition of $\mathcal{DM}_{pk}$, the resulting RDF graph will have an inconsistent triple TRIPLE(a, "owl:differentFrom", a), which would generate an inconsistency under the OWL 2 DL semantics.

B.5 Proof of Theorem 3

For the sake of contradiction, assume that $\mathcal{M}$ is a monotone and semantics preserving direct mapping. Then consider a schema $R$ containing at least two distinct relation names $R_1$, $R_2$, and consider a set $\Sigma$ of PKs and FKs over $R$ containing at least one foreign key from $R_1$ to $R_2$. Then we have that there exist instances $I_1$, $I_2$ of $R$ such that $I_1 \subseteq I_2$, $I_1$ does not satisfy $\Sigma$ and $I_2$ does satisfy $\Sigma$. Given that $\mathcal{M}$ is semantics preserving, we know that $\mathcal{M}(R, \Sigma, I_2)$ is consistent under the OWL 2 DL semantics, while $\mathcal{M}(R, \Sigma, I_1)$ is not. Given that $\mathcal{M}$ is monotone, we have that $\mathcal{M}(R, \Sigma, I_1) \subseteq \mathcal{M}(R, \Sigma, I_2)$. But then we conclude that $\mathcal{M}(R, \Sigma, I_1)$ is also consistent under the OWL 2 DL semantics, given that $\mathcal{M}(R, \Sigma, I_2)$ is consistent and $\mathcal{M}(R, \Sigma, I_1) \subseteq \mathcal{M}(R, \Sigma, I_2)$, which leads to a contradiction.
B.6 Proof of Theorem 4

It is straightforward to see that given a relational schema $R$, set $\Sigma$ of PKs and FKS over $R$ and instance $I$ of $R$ such that $I \models \Sigma$, it holds that $DM_{pk+fk}(R, \Sigma, I)$ is consistent under the OWL 2 DL semantics. Likewise, if $I \not\models \Sigma$, then by definition of $DM_{pk+fk}$, the resulting RDF graph will contain an inconsistent triple $\text{TRIPLE}(a, \text{"owl:differentFrom"}, a)$, which would generate an inconsistency under the OWL 2 DL semantics.