Transforming OWL 2 RL Schemas to Relational Schemas with Open-world or Closed-world Semantics

Lama Al Khuzayem
Data and Knowledge Engineering Group
Department of Computing
Imperial College London
laa11@imperial.ac.uk

Yu Liu
Data and Knowledge Engineering Group
Department of Computing
Imperial College London
yu.liu11@imperial.ac.uk

Peter McBrien
Data and Knowledge Engineering Group
Department of Computing
Imperial College London
p.mcbrien@imperial.ac.uk

ABSTRACT

Storing and processing Semantic Web knowledge in **relational database management systems (RDBMSs)** is currently a growing interest in both academia and industry. In this paper, we present OWLRel, a database-driven ontology reasoner that supports sound reasoning of OWL 2 RL and can adhere to either the open-world assumption or the closed-world assumption. This is achieved by regarding the mapping of OWL 2 RL schemas to relational schemas as a two-phase process: (1) Convert the OWL 2 RL model into a logical relational model where the operational semantics of constraints are not specified. (2) Implement the OWL 2 RL constraints in the relational model either as triggers achieving an **open-world semantics (OWS)** approach, or as constraints achieving a **closed-world semantics (CWS)** approach.

Keywords

Ontologies, OWL 2 RL, Relational Databases, Database Triggers, Transactional Reasoning, BAV Transformations.

1. INTRODUCTION

With the increasing number of ontologies available on the web, a growing interest has arisen for developing systems that not only store ontologies expressed in the **web ontology language** (**OWL**) [25] in a **relational database management system** (**RDBMS**), but also perform reasoning over the instances to capture the **open-world semantics** (**OWS**) characteristics of ontologies [8, 27, 18]. Hence, the problem which is the focus of this paper, can be broken down into two parts. Firstly, how can we map the OWL schemas to relational schemas, and secondly how to perform reasoning over the instances held in this relational schema.

Most of the state-of-the-art proposals perform the transformation from knowledge model to data model at a high-level, which is often referred to as **direct mapping** (**DM**). This involves specifying specific mappings from constructs in OWL to constructs in the relational model, or *vice versa*. By contrast, in this paper, we show how to translate schemas expressed in OWL 2 RL (a profile of the most recent version of the web ontology language OWL 2 [26] which aims for rule-based implementations) into relational schemas via an intermediary low-level **hypergraph data model** (**HDM**) [19].

Using the HDM as an intermediate language has several advantages. First is the benefit of abstracting the high-level constructs in OWL and the relational model to a set of core low-level elemental modelling primitives (nodes, edges, and constraints) [21], making apparent what are the precise differences in the logical semantics of the OWL and relational models. The second benfit is that it has already been shown how to use the HDM to translate betweem ER, ORM, relational, UML and XML modelling languages [4, 13], and hence the work mapping OWL to relational models presented in this paper will transitivly also allow OWL to be mapped to other data models. Thirdly, the **intermodel transformations** described in [4] expressed as BAV mappings [14] are based on using five types of HDM equivalence rules that transform one HDM schema into an equivalent HDM schema, and these equivalence rules may be directly applied to the new task of mapping between the OWL and relational models.

RDBMSs provide a means of storage for ontologies that is able to process ontologies with large number of individuals in a manner more efficient than using Tableaux-based reasoners such as Pellet [20] and Fact++ [23]. A drawback, however, is that we are forced to make the **unique name assumption (UNA)**. A challange we meet in this paper is that normal semantics of an RDBMS is **closed-world semantics** (**CWS**), but that of OWL is OWS. We meet this challange by building a single framework that maps the OWL OWS to either an equivalent OWS in the RDBMS, or a CWS in the RDBMS.

In general, the process of reasoning may be broken down into: classification of the **terminology box** (**T-Box**) and type-inference over the **assertion box** (**A-Box**). The classification of the T-Box is not the concern of this paper, and our implementation work has relied on an OWL reasoner, Pellet, to perform this task. Type-inference, in the context of databases, is the process of relating values for tables/-columns to values in other tables/columns. How this is executed will depend on the choice of OWS or CWS. Considering the **description logic** (**DL**) [3] rule Father ≡ Man ⊓ Parent, in an OWS, we could deduce the following inference rules:

1. Inserting John as an instance of Father would then cause John to be inserted into both Man and Parent if not already present.
2. Inserting John into Man would also cause John to be inserted into Father if John was already in Parent.
3. Inserting John into Parent would also cause John to be inserted into Father if John was already in Man.

By contrast, in a CWS:

1. An insert of John into Father would fail unless John was already an instance of both Man and Parent.
2. An insert of John into Man would fail if John was already present in Parent but not present in Father.
3. An insert of John into Parent would fail if John was already present in Man but not present in Father.

A further objective of reasoning in an RDBMS is to achieve **transactional reasoning** [16], where the results of reasoning derived from data changed by the database operations should be available as part of the atomic action of the transaction. To achieve transactional reasoning, two approaches were identified in [16]: **view-based reasoning** (**VBR**), where rules are used to derive the result of reasoning as each query is executed over the database, and **trigger-based reasoning** (**TBR**) where triggers (active rules) are used to materialise the result of reasoning at data insertion time, and queries simply read the materialised views. The advantages and disadvantages of using views or materialised views are well known, and each serve different real world requirements. Specific advantages of the TBR approach include that it is very fast at query processing, and that reasoning is incremental, since insertion of data only requires the change to reasoned results to be computed.

To illustrate the concept of transactional reasoning, let us consider again the DL rule Father ≡ Man ⊓ Parent. If John was already recorded as a Man, then if a transaction added that John was a Parent, then any query on the result of the transaction should also be able to view that John is a Father. However, most approaches for storing ontology data in an RDBMS will make the process of reasoning be detached from transaction processing of the data. Specifically, in our example, after the transaction that added John was a Parent, the database could be queried and find that he was not a Father until a separate process of reasoning had derived that fact. Hence, we would say that these approaches do not support transactional reasoning.

The novelty of this paper, which distinguishes it from previous work in the area (reviewed in Section 11) can be summarised as follows:

1. We represent OWL 2 RL axioms and constructs in the HDM.
2. We produce lossless transformations of OWL 2 RL knowledge bases to relational schemas. The trasnsformations are lossless in the sense that all the semantics of the OWL 2 RL model are presented in the relational model.
3. Expressing the transformations as BAV mappings means we have a precise defintion of the equivalence between schemas, and can map data back and forth between the OWL 2 RL and relational schemas.
4. We provide a single mapping approach for handling the axioms of OWL 2 RL schemas which can result in either CWS or OWS in the RDBMS.
5. We support sound transactional reasoning of OWL 2 RL ontologies with large A-Boxes.

The remainder of this paper is structured as follows. In Section 2 we detail OWLRel's architecture and in Section 3 we review the HDM. In Section 4, we show the complete representations of OWL 2 RL constructs in HDM and provide a transformation example followed by explaining the intermodel transformations in Section 5. The process of mapping the resulting HDM schema to an RDBMS is explained in Section 6. We present our novel approach of transforming OWL 2 RL constructs to relational schemas via the HDM under OWS and CWS in Sections 7 and 8 respectively. Section 9 highlights the operational semantics differences between OWS and CWS. We show our system's evaluation results in Section 10 and review related work in the area in Section 11. Finally, in Section 12, we state our conclusions and elaborate on future directions.

2. OWLREL ARCHITECTURE

OWLRel exploits this division of the OWL reasoning process: classification of the T-Box and type-inference of the A-Box to build a reasoning system in several steps, for which the overall design is illustrated in Figure 1. The process is as follows:

1 OWLRel uses the OWL API [7] to load an OWL ontology file and then separates the T-Box and the A-Box.

2 The T-Box is passed into a reasoner for classification. Since this step is conducted only once as a process of building up the database schema, the objective is to have as complete reasoning results as possible. The OWLRel system supports any reasoner that works with OWL API, and our evaluation results in Section 10 are based on using Pellet.

3 The fully classified T-Box is mapped to a HDM schema called the HDM-OWL2RL schema. The BAV mapping approach used in this transformation describes the the mappings between schemas on a construct by construct basis, as a pathway of primitive transformation steps applied in sequence. Details of this transformation are found in Section 4.

circled4 OWLRel then applies BAV equivalence rules on the HDM-OWL2RL schema to produce an equivalent HDM schema called HDM-OWL2RL+rel. The purpose of this step is to prepare the HDM schema which will be mapped to a relational model. Details can be found in in Section 5.

5 A core relational database schema that resembles part of the HDM-OWL2RL+rel schema is created in an RDBMS which contains only tables, columns, and primary keys as discussed in Section 6. The rest of the HDM constraints are treated separately under OWS and CWS as in the following two alternative steps.

6 Under OWS, OWLRel generates a set of SQL statements to create triggers for the HDM constraints. The OWLRel system uses an external file of template triggers to generate

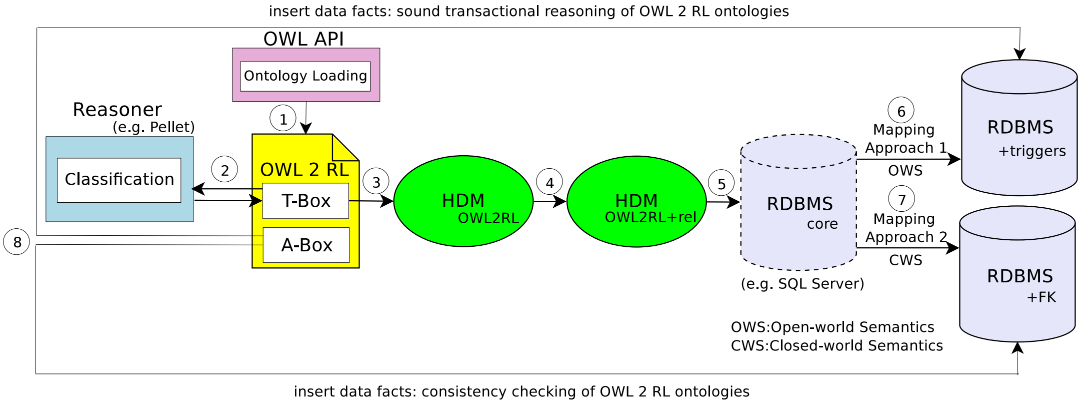


Figure 1: OWL 2 RL to Relational Mapping Approaches

the SQL triggers from the logical triggers which are presented in Section 7.
(7) Under CWS, OWLRel creates SQL constraints for those unconverted constraints from HDM-OWL2RL+rel as explained in Section 8.
(8) Once the SQL schema plus triggers or constraints have been created, the A-Box reasoning in an OWS context can take place by inserting data using SQL statements, which results in invoking the triggers that perform type inference and insert derived data into the database. If an inconsistency arises from a particular reasoning step, the transaction in which it occurs will be rolled back by removing any reasoned data as well as the original inserted data. In a CWS scenario, as the A-Box is being inserted to the database, constraints will check the consistency of the database. Hence, any data that violates these constraints will be rejected, rather than performing type inference.

The result of these steps is a stand-alone fully type inferencing system in an RDBMS under **OWS**, which produces **sound reasoning** for ontologies expressed in the OWL 2 RL profile or ensuring the **consistency** of the ontologies in a **CWS** setting.

3. HDM OVERVIEW

In this Section, we provide a brief review of the HDM defined in [4, 19]. An HDM schema S is a triple $\langle Nodes, Edges, Cons\rangle$ where:

- $Nodes$ is a set of nodes in the graph such that $Nodes \subseteq \{\mathsf{node}{:}\langle\langle \mathsf{n_n}\rangle\rangle \mid n_n \in Names\}$. Given data instance I of the HDM schema, the function $Ext_I(\mathsf{node}{:}\langle\langle \mathsf{n_n}\rangle\rangle)$ gives the set of data values associated with $\mathsf{node}{:}\langle\langle \mathsf{n_n}\rangle\rangle$.
- $Schemes = Nodes \cup Edges$
- $Edges$ is a set of edges in the graph such that $Edges \subseteq \{\mathsf{edge}{:}\langle\langle \mathsf{n_e}, \mathsf{s_1}, \ldots, \mathsf{s_n}\rangle\rangle \mid n_e \in Names \cup \{\_\} \wedge s_1 \in Schemes \wedge \ldots \wedge s_n \in Schemes\}$. Note that this defines a hypergraph edge since edges can connect more than two nodes, and also defines a nested graph, since edges can also connect to other edges as follows:
$\forall I.Ext_I(\mathsf{edge}{:}\langle\langle \mathsf{n_e}, \mathsf{s_1}, \ldots, \mathsf{s_n}\rangle\rangle) \subseteq Ext_I(S_1)\times \ldots Ext_I(S_n)$
- $Cons \subseteq \{c(s_1, \ldots, s_n) \mid c \in Funcs \wedge s_1 \in Schemes \wedge \ldots \wedge s_n \in Schemes\}$ $Cons$ is a set of boolean-valued functions (constraints) whose variables are members of $Schemes$ and where the set of functions $Funcs$ forms the HDM constraint language. The set of constraints used in this paper are as follows:
  - $\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{s_1}, \mathsf{s_2}\rangle\rangle$ is the **inclusion** constraint which states that scheme s_1 is always a subset of scheme s_2: $\forall I.Ext_I(s_1) \subseteq Ext_I(s_2)$
  - $\mathsf{cons}{:}\langle\langle \not\cap, \mathsf{s_1}, \ldots, \mathsf{s_n}\rangle\rangle$ is the **exclusion** constraint which states that all the associate schemes are disjoint from each other:
$\forall I.1 \leq x \leq n, 1 \leq y \leq n, x \neq y.Ext_I(s_x) \cap Ext_I(s_y) = \emptyset$
  - $\mathsf{cons}{:}\langle\langle \cup, \mathsf{s_1}, \ldots, \mathsf{s_n}, \mathsf{s}\rangle\rangle$ is the **union** constraint stating scheme s is the union of schemes $s_1, \ldots, s_n$:
$\forall I.Ext_I(s) = Ext_I(s_1) \cup \ldots \cup Ext_I(s_n)$
  - $\mathsf{cons}{:}\langle\langle \rhd, \mathsf{s_1}, \ldots, \mathsf{s_m}, \mathsf{s}\rangle\rangle$ is the **mandatory** constraint stating that every combination of values that appears in schemes $s_1, \ldots, s_m$ must appear in the edge s connecting those schemes.
  - $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{s_1}, \ldots, \mathsf{s_m}, \mathsf{s}\rangle\rangle$ is the **unique** constraint stating that every combination of values that appears in schemes $s_1, \ldots, s_m$ must appear no more than once in the edge s connecting those schemes.
  - $\mathsf{cons}{:}\langle\langle \xrightarrow{\mathrm{id}}, \mathsf{s_1}, \mathsf{s}\rangle\rangle$ is the **reflexive** constraint, stating that for any value in s_1 must appear reflexivly in the edge s that connects to s_1, so that $\forall I.Ext_I(s_1) \times Ext_I(s_1) \subseteq Ext_I(s)$
  - $\mathsf{cons}{:}\langle\langle |1|, \mathsf{s}\rangle\rangle$ is the **instance** constraint, stating that only one value may be stored in s such that $\forall I.|Ext_I(s)| = 1$

  In addition to referring to schemes directly, constraints may also take joins and projections of schemes as arguments.

To illustrate the use of HDM to describe the semantics of high-level data models, consider the relational schema in Example 3.1 (where primary keys are underlined, and nullable column names are suffixed by a question mark). We will later show how this schema can be derived from the OWL 2 RL ontology listed in Figure 2.

EXAMPLE 3.1. **Family Relational Database Schema**

Person(id, spouse?)
Man(id, wife?)
Woman(id, husband?)
Parent(id)
JohnsChildren(id)
hasChild(parent, child)
hasParent(child, parent)
hasGrandParent(grandchild, grandparent)
hasAncestor(descendant, ancestor)

□

Using the approach from [12], we can translate the relational schema into a HDM schema as follows. For each relational table we create a node, and connect it via HDM edges to nodes created for each column. Hence for the Woman table we create:

node:⟪Woman⟫ edge:⟪_,Woman,woman:id⟫
node:⟪woman:id⟫ edge:⟪_,Woman,husband⟫
node:⟪husband⟫

Since values in columns cannot exist in isolation from rows in the table, we state that the nodes representing columns have a mandatory association with the edge connecting them to the nodes representing the table:

cons:⟪$\rhd$, node:⟪woman:id⟫, edge:⟪_, Woman, woman:id⟫⟫
cons:⟪$\rhd$, node:⟪husband⟫, edge:⟪_, Woman, husband⟫⟫

and since each column of a relation may take only one value, it follows that the node representing table is connected to the same edges by a unique constraint:

cons:⟪$\lhd$, node:⟪Woman⟫, edge:⟪_, Woman, woman:id⟫⟫
cons:⟪$\lhd$, node:⟪Woman⟫, edge:⟪_, Woman, husband⟫⟫

If columns are not nullable, such as woman.id, we also state that the association of the table to the column edge is mandatory:

cons:⟪$\rhd$, node:⟪Woman⟫, edge:⟪_, Woman, woman:id⟫⟫

and if the column is key, then we state that the same edge is reflexive:

cons:⟪$\xrightarrow{\text{id}}$, node:⟪Woman⟫, edge:⟪_, Woman, woman:id⟫⟫

which in combination with the unique and mandatory constraints has the consequence that the extent of the node representing the table becomes the set of key values of the table.

Note that you can view the HDM representation of the relational model (HDM nodes connected by HDM edges) as a forest of two-level trees, where the root of each tree is a node representing a table, the leafs of the tree are nodes representing columns, and HDM inclusion constraints exist only between leaf nodes. Other types of HDM constraints may be associated with the edges between nodes.

4. REPRESENTING OWL 2 RL IN HDM

OWL 2 RL is a syntactic subset of OWL 2 DL [26], and OWL 2 DL is an implementation of the DL $\mathcal{SROIQ}(\mathcal{D})$ with keys added. OWL 2 RL supports almost all OWL 2 axioms except for reflexive object properties and disjoint union expressions, and restricts the usage of some class expressions to make it possible to reason using rule-based engines with a complexity of **PTIME-complete**. Thus, it is well suited for applications that require scalable reasoning without loosing too much expressivity.

We now discuss how OWL 2 RL constructs and axioms may be represented in HDM which corresponds to Step 3 in Figure 1. For conciseness, we only discuss some of those OWL 2 RL constructs listed in Tables 1 and 2, which are sufficient to describe how the OWL 2 RL knowledge base illustrated in Figure 2 can be translated into a HDM schema depicted in Figure 3.

All OWL classes are represented as HDM nodes. For example, class Person is represented as: node:⟪Person⟫

Object properties are represented as HDM edges with different HDM constraints depending on the type of the objectProperty. For example, the hasSpouse property is both a symmetricProperty and a functionalProperty as denoted in rules (26) and (27). In HDM we represent it as follows:

edge:⟪hasSpouse, Person, Person⟫
cons:⟪ $\subseteq, \pi_{\langle \text{Person\#2,Person\#1} \rangle}$ ⟪hasSpouse, Person, Person⟫,
⟪hasSpouse, Person, Person⟫⟫
cons:⟪$\lhd$, Person#1, ⟪hasSpouse, Person, Person⟫⟫

Note in the above rule, where the edge hasSpouse links node ⟪Person⟫ with itself, we can disambiguate the first and second occurrence of ⟪Person⟫ with #1 and #2.

All OWL 2 RL axioms are represented as HDM constraints. For example, the subClassOf and subPropertyOf axioms denoted in rules (1) and (21) are represented as inclusion constraints ($\subseteq$) where the first element is subsumed by the second as follows:

cons:⟪ $\subseteq$, Man, Person⟫
cons:⟪ $\subseteq$, ⟪hasHusband, Woman, Man⟫,
⟪hasSpouse, Person, Person⟫⟫

If a class is a complementOf another class as given in rule (3) that is represented as an exclusion constraint ($\not\cap$) between the two classes and the union ($\cup$) of the two classes gives you the class Thing as follows:

cons:⟪$\not\cap$, Man, Woman⟫
cons:⟪$\cup$, Thing, Man, Woman⟫

The someValuesFrom construct denoted in rule (4) is resembeled in HDM as a node connected to the filler node (i.e. class Person) with an edge and a mandatory ($\rhd$) constraint and making the newly created edge subset of the edge resembling the same property as follows:

node:⟪∃hasChild.Person⟫
edge:⟪hasChildIE, ∃hasChild.Person, Person⟫
cons:⟪$\rhd$, ∃hasChild.Person,
⟪hasChildIE, ∃hasChild.Person, Person⟫⟫
cons:⟪ $\subseteq$, ⟪hasChildIE, ∃hasChild.Person, Person⟫,
⟪hasChild, Person, Person⟫⟫

The hasValues construct denoted in rule (5) is represented by a node for the individual John with an attached cardinality constraint to 1 as follows:

node:⟪John⟫
cons:⟪|1|, John⟫

Then, similar to representing ∃ hasChild.Person, we create a node ⟪∃hasParent.John⟫ and connect it via an edge to the node ⟪John⟫ and make it mandatory on the edge. We then represent the equivalentClass construct denoted in the same rule by two inclusion constraints ($\subseteq$) in both directions to make the first class subset of the second, and the second subset of the first as follows:

cons:⟪ $\subseteq$, ∃hasParent.John, JohnsChildren⟫
cons:⟪ $\subseteq$, JohnsChildren, ∃hasParent.John⟫

The fact that one property is the inverseOf another as shown in rule (24) is represented by an inclusion constraint between the property and the projection (π) of its inverse in the reverse order as follows:

cons:⟪ $\subseteq, \pi_{\langle \text{hasHusband,Man,Woman} \rangle}$, ⟪hasWife, Man, Woman⟫⟫
cons:⟪ $\subseteq, \pi_{\langle \text{hasWife,Woman,Man} \rangle}$, ⟪hasHusband, Woman, Man⟫⟫

$$\mathsf{Man} \sqsubseteq \mathsf{Person} \tag{1}$$
$$\mathsf{Woman} \sqsubseteq \mathsf{Person} \tag{2}$$
$$\mathsf{Man} \equiv \neg\mathsf{Woman} \tag{3}$$
$$\exists\ \mathsf{hasChild}.\mathsf{Person} \sqsubseteq \mathsf{Parent} \tag{4}$$
$$\mathsf{JohnsChildren} \equiv \exists\ \mathsf{hasParent}.\{\mathsf{John}\} \tag{5}$$
$$\top \sqsubseteq \forall\ \mathsf{hasChild}.\mathsf{Person} \tag{6}$$
$$\top \sqsubseteq \forall\ \mathsf{hasChild}^{-}.\mathsf{Person} \tag{7}$$
$$\top \sqsubseteq \forall\ \mathsf{hasHusband}.\mathsf{Man} \tag{8}$$
$$\top \sqsubseteq \forall\ \mathsf{hasHusband}^{-}.\mathsf{Woman} \tag{9}$$
$$\top \sqsubseteq \forall\ \mathsf{hasWife}.\mathsf{Woman} \tag{10}$$
$$\top \sqsubseteq \forall\ \mathsf{hasWife}^{-}.\mathsf{Man} \tag{11}$$
$$\top \sqsubseteq \forall\ \mathsf{hasSpouse}.\mathsf{Person} \tag{12}$$
$$\top \sqsubseteq \forall\ \mathsf{hasSpouse}^{-}.\mathsf{Person} \tag{13}$$
$$\top \sqsubseteq \forall\ \mathsf{hasParent}.\mathsf{Person} \tag{14}$$
$$\top \sqsubseteq \forall\ \mathsf{hasParent}^{-}.\mathsf{Person} \tag{15}$$
$$\top \sqsubseteq \forall\ \mathsf{hasAncestor}.\mathsf{Person} \tag{16}$$
$$\top \sqsubseteq \forall\ \mathsf{hasAncestor}^{-}.\mathsf{Person} \tag{17}$$
$$\top \sqsubseteq \forall\ \mathsf{hasGrandParent}.\mathsf{Person} \tag{18}$$
$$\top \sqsubseteq \forall\ \mathsf{hasGrandParent}^{-}.\mathsf{Person} \tag{19}$$
$$\mathsf{hasWife} \sqsubseteq \mathsf{hasSpouse} \tag{20}$$
$$\mathsf{hasHusband} \sqsubseteq \mathsf{hasSpouse} \tag{21}$$
$$\mathsf{hasParent} \sqsubseteq \mathsf{hasGrandParent} \tag{22}$$
$$\mathsf{hasGrandParent} \sqsubseteq \mathsf{hasAncestor} \tag{23}$$
$$\mathsf{hasHusband} \equiv \mathsf{hasWife}^{-} \tag{24}$$
$$\mathsf{hasParent} \circ \mathsf{hasParent} \equiv \mathsf{hasGrandParent} \tag{25}$$
$$\mathsf{hasSpouse}^{-} \sqsubseteq \mathsf{hasSpouse} \tag{26}$$
$$\top \sqsubseteq \leq 1\ \mathsf{hasSpouse} \tag{27}$$
$$\top \sqsubseteq \leq 1\ \mathsf{hasHusband} \tag{28}$$
$$\top \sqsubseteq \leq 1\ \mathsf{hasWife} \tag{29}$$
$$\mathsf{hasAncestor} \circ \mathsf{hasAncestor} \sqsubseteq \mathsf{hasAncestor} \tag{30}$$

(a) The T-Box of the OWL 2 RL Family Knowledge Base

$$\mathsf{Man}(\mathsf{John}) \tag{31}$$
$$\mathsf{Woman}(\mathsf{Mary}) \tag{32}$$
$$\mathsf{hasAncestor}(\mathsf{Michael}, \mathsf{Alex}) \tag{33}$$
$$\mathsf{hasHusband}(\mathsf{Mary}, \mathsf{John}) \tag{34}$$
$$\mathsf{hasParent}(\mathsf{Lewis}, \mathsf{Albert}) \tag{35}$$
$$\mathsf{hasParent}(\mathsf{Albert}, \mathsf{Alex}) \tag{36}$$

(b) The A-Box of the OWL 2 RL Family Knowledge Base

Figure 2: A Family Knowledge Base Expressed in OWL 2 RL

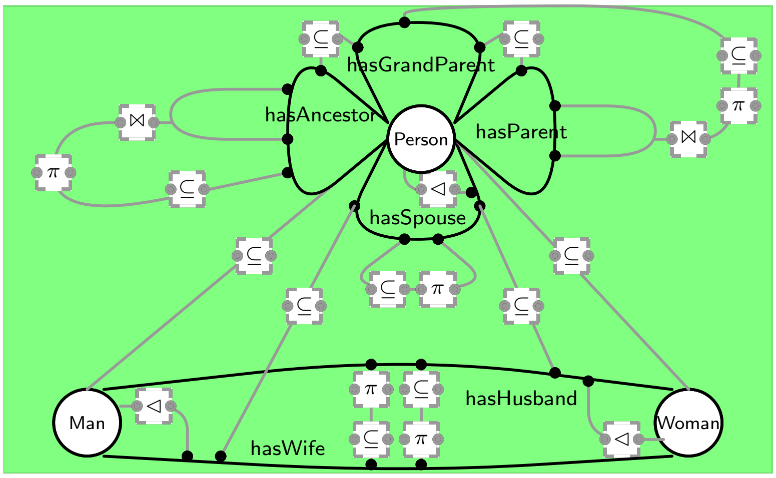


Figure 3: HDM Representation of the OWL 2 RL Family T-Box Rules (1-2) & (8-30)

A propertyChain such as the one given in rule (25) is represented by making the projection (π) of joining ($\bowtie$) the subproperties subset of itself as follows:

edge: $\langle\langle \mathsf{hasGrandParent}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle$

cons: $\langle\langle \subseteq, \pi_{\langle \mathsf{Person\#1}, \mathsf{Person\#4} \rangle} \langle\langle \mathsf{hasParent}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle$
$\bowtie_{\mathsf{Person\#2}=\mathsf{Person\#3}} \langle\langle \mathsf{hasParent}, \mathsf{Person\#3}, \mathsf{Person\#4} \rangle\rangle,$
$\langle\langle \mathsf{hasGrandParent}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle \rangle\rangle$

Similarly, a transitiveProperty such as the one denoted in rule (30) is represented as follows:

edge: $\langle\langle \mathsf{hasAncestor}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle$

cons: $\langle\langle \subseteq, \pi_{\langle \mathsf{Person\#1}, \mathsf{Person\#4} \rangle} \langle\langle \mathsf{hasAncestor}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle$
$\bowtie_{\mathsf{Person\#2}=\mathsf{Person\#3}} \langle\langle \mathsf{hasAncestor}, \mathsf{Person\#3}, \mathsf{Person\#4} \rangle\rangle,$
$\langle\langle \mathsf{hasAncestor}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle \rangle\rangle$

Note that in the HDM diagram, HDM nodes are represented by white circles with thick outlines, and HDM edges are represented by thick black lines. The HDM constraint language is represented by grey dashed boxes connected by grey lines to the nodes and edges to which the constraint applies. Edges pass through black circles in a straight line, hence any edge or constraint applying to an edge meets that edge at an angle.

5. HDM TRANSFORMATIONS

Step (4) in Figure 1 performs a type of normalisation on the HDM-OWL2RL schema, to produce an equivalent HDM-OWL2RL+rel schema that can be directly mapped into a relational schema. This normalisation process is important to overcome the fundamental differences between the two modelling languages. On the one hand, OWL 2 RL is a knowledge model that has the notion of classes, properties. On the other hand, relational, is a key-based data model that has tables, columns, primary key (PK) and foreign key (FK) constraints.

Our three step process converts the HDM graph representing OWL 2 RL into an equivalent graph (in terms of information capacity) that can then be mapped into a relational schema. This uses a set of HDM equivalence mappings presented in [4].

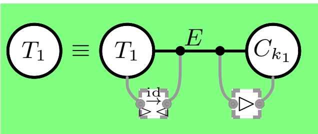
(A) Transform the implied object identifiers (OID) of OWL classes into explicit keys to form PK columns for their respective tables in the relational model.

This can be achieved using a HDM graph equivalence

Table 1: HDM Representations for OWL 2 RL Class Axioms

OWL 2 Construct	DL Syntax	HDM Representation
Class	C	node:$\langle\langle \mathsf{C} \rangle\rangle$
subClassOf	$\mathsf{C} \sqsubseteq \mathsf{D}$	cons:$\langle\langle \subseteq, \mathsf{C}, \mathsf{D} \rangle\rangle$
equivalentClass	$\mathsf{C} \equiv \mathsf{D}$	cons:$\langle\langle \subseteq, \mathsf{C}, \mathsf{D} \rangle\rangle$, cons:$\langle\langle \subseteq, \mathsf{D}, \mathsf{C} \rangle\rangle$
classDisjointWith	$\mathsf{C} \sqcap \mathsf{D} \sqsubseteq \perp$	cons:$\langle\langle \not\cap, \mathsf{C}, \mathsf{D} \rangle\rangle$
complementOf	$\mathsf{C} \equiv \neg\mathsf{D}$	cons:$\langle\langle \not\cap, \mathsf{C}, \mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{C}, \mathsf{D}, \mathsf{Thing} \rangle\rangle$
allValuesFrom	$\forall\mathsf{P.D}$	node:$\langle\langle \neg\mathsf{D} \rangle\rangle$, cons:$\langle\langle \not\cap, \mathsf{D}, \neg\mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{D}, \neg\mathsf{D}, \mathsf{Thing} \rangle\rangle$ node:$\langle\langle \exists\mathsf{P}.\neg\mathsf{D} \rangle\rangle$, edge:$\langle\langle \mathsf{P\_IE}, \exists\mathsf{P}.\neg\mathsf{D}, \neg\mathsf{D} \rangle\rangle$, cons:$\langle\langle \rhd, \exists\mathsf{P}.\neg\mathsf{D}, \langle\langle \mathsf{P\_IE}, \exists\mathsf{P}.\neg\mathsf{D}, \neg\mathsf{D} \rangle\rangle \rangle\rangle$ node:$\langle\langle \forall\mathsf{P.D} \rangle\rangle$, cons:$\langle\langle \not\cap, \forall\mathsf{P.D}, \exists\mathsf{P}.\neg\mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \forall\mathsf{P.D}, \exists\mathsf{P}.\neg\mathsf{D}, \mathsf{Thing} \rangle\rangle$
someValuesFrom	$\exists\mathsf{P.D}$	node:$\langle\langle \exists\mathsf{P.D} \rangle\rangle$, edge:$\langle\langle \mathsf{P\_IE}, \exists\mathsf{P.D}, \mathsf{D} \rangle\rangle$ cons:$\langle\langle \rhd, \exists\mathsf{P.D}, \langle\langle \mathsf{P\_IE}, \exists\mathsf{P.D}, \mathsf{D} \rangle\rangle \rangle\rangle$, cons:$\langle\langle \subseteq, \langle\langle \mathsf{P\_IE}, \exists\mathsf{P.D}, \mathsf{D} \rangle\rangle, \langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D} \rangle\rangle \rangle\rangle$
hasValue	$\exists\mathsf{P}.\{\mathsf{a}\}$	node:$\langle\langle \exists\mathsf{P.a} \rangle\rangle$, edge:$\langle\langle \mathsf{P\_IE}, \exists\mathsf{P.a}, \mathsf{a} \rangle\rangle$ cons:$\langle\langle \rhd, \exists\mathsf{P.a}, \langle\langle \mathsf{P\_IE}, \exists\mathsf{P.a}, \mathsf{a} \rangle\rangle \rangle\rangle$, cons:$\langle\langle \subseteq, \langle\langle \mathsf{P\_IE}, \exists\mathsf{P.a}, \mathsf{a} \rangle\rangle \langle\langle \mathsf{P}, \mathsf{D}, \mathsf{D} \rangle\rangle \rangle\rangle$
oneOf	$\{\mathsf{a1}, \ldots, \mathsf{an}\}$	node:$\langle\langle \mathsf{a_1\_a_i\_a_n} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{a}_1, \ldots, \mathsf{a}_n, \mathsf{a_1\_a_i\_a_n} \rangle\rangle$
maxCardinality	$\leq\mathsf{nP}$	node:$\langle\langle \leq \mathsf{nP} \rangle\rangle$, edge:$\langle\langle \mathsf{P\_IE}, \leq \mathsf{nP}, \mathsf{D} \rangle\rangle$, cons:$\langle\langle \rhd^{\mathsf{n}}, \leq \mathsf{nP}, \langle\langle \mathsf{P\_IE}, \leq \mathsf{nP}, \mathsf{D} \rangle\rangle \rangle\rangle$
intersectionOf	$\mathsf{C} \sqcap \mathsf{D}$	node:$\langle\langle \mathsf{C} - \mathsf{D} \rangle\rangle$, cons:$\langle\langle \subseteq, \mathsf{C} - \mathsf{D}, \mathsf{C} \rangle\rangle$, node:$\langle\langle \mathsf{C} \cup \mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{C} - \mathsf{D}, \mathsf{D}, \mathsf{C} \cup \mathsf{D} \rangle\rangle$ cons:$\langle\langle \cup, \mathsf{C}, \mathsf{D}, \mathsf{C} \cup \mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{C} - \mathsf{D}, \mathsf{D} \rangle\rangle$, node:$\langle\langle \mathsf{C} \cap \mathsf{D} \rangle\rangle$, cons:$\langle\langle \subseteq, \mathsf{C} \cap \mathsf{D}, \mathsf{C} \rangle\rangle$ cons:$\langle\langle \not\cap, \mathsf{C} \cap \mathsf{D}, \mathsf{C} - \mathsf{D} \rangle\rangle$, cons:$\langle\langle \cup, \mathsf{C} \cap \mathsf{D}, \mathsf{C} - \mathsf{D}, \mathsf{C} \rangle\rangle$
namedIndividual	a	node:$\langle\langle \mathsf{a} \rangle\rangle$, cons:$\langle\langle \lvert 1 \rvert, \mathsf{a} \rangle\rangle$

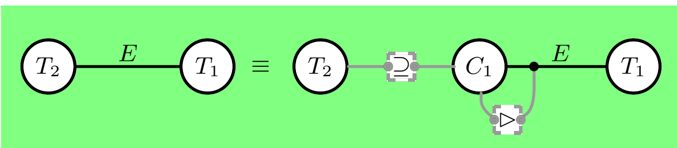
transformation, where the function: $\mathsf{inverse\_identity\_node\_merge}(\langle\langle \mathsf{T_1} \rangle\rangle, \langle\langle \mathsf{C_{k_1}} \rangle\rangle)$ will take an existing node $\langle\langle \mathsf{T_1} \rangle\rangle$, and creates a new node $\langle\langle \mathsf{C_{k_1}} \rangle\rangle$ connected to $\langle\langle \mathsf{T_1} \rangle\rangle$ by an edge. The edge has constraints that ensure that each instance of $\langle\langle \mathsf{T_1} \rangle\rangle$ appears at least once ($\lhd$), at most once ($\rhd$) and reflexively ($\overset{\text{id}}{\to}$) in the edge, so that the contents of $\langle\langle \mathsf{C_{k_1}} \rangle\rangle$ must be identical to $\langle\langle \mathsf{T_1} \rangle\rangle$. This is illustrated by:



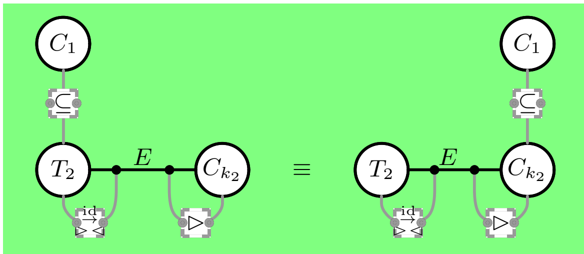
Applying this step on the HDM OWL 2 RL schema in Figure 3 would be achieved by:
$\mathsf{inverse\_identity\_node\_merge}(\langle\langle \mathsf{Person} \rangle\rangle, \langle\langle \mathsf{Person{:}id} \rangle\rangle)$
$\mathsf{inverse\_identity\_node\_merge}(\langle\langle \mathsf{Woman} \rangle\rangle, \langle\langle \mathsf{Woman{:}id} \rangle\rangle)$
$\mathsf{inverse\_identity\_node\_merge}(\langle\langle \mathsf{Man} \rangle\rangle, \langle\langle \mathsf{Man{:}id} \rangle\rangle)$
and would result in HDM constructs for tables and their keys as described in Section 3.

(B) Convert HDM edges representing OWL properties between OWL classes into the HDM equivalent of a column with a foreign key.

This process involves using two HDM equivalences. The first equivalence mapping: $\mathsf{inverse\_inclusion\_merge}(\langle\langle \mathsf{E}, \mathsf{T_1}, \mathsf{T_2} \rangle\rangle, \langle\langle \mathsf{C_1} \rangle\rangle)$ uses the $\langle\langle \mathsf{E}, \mathsf{T_1}, \mathsf{T_2} \rangle\rangle$ edge to identify those members of $\langle\langle \mathsf{T_1} \rangle\rangle$ that participate in the edge, and put them in a new node $\langle\langle \mathsf{C_1} \rangle\rangle$ that is a subset of $\langle\langle \mathsf{T_2} \rangle\rangle$ as illustrated below.



Once this equivalence mapping has been performed, the subset $\langle\langle \subseteq, \langle\langle \mathsf{T_1} \rangle\rangle, \langle\langle \mathsf{C_1} \rangle\rangle \rangle\rangle$ is used in another mapping to be redirected to node $\langle\langle \mathsf{C_{k_2}} \rangle\rangle$ representing the key of table represented by $\langle\langle \mathsf{T_2} \rangle\rangle$. This is generated by a second equivalence mapping $\mathsf{redirect\_inclusion\_constraint}(\langle\langle \subseteq, \langle\langle \mathsf{T_1} \rangle\rangle, \langle\langle \mathsf{C_1} \rangle\rangle \rangle\rangle, \langle\langle \mathsf{C_{k_2}} \rangle\rangle)$



We can apply these two equivalences to the edge: $\langle\langle \mathsf{hasSpouse}, \mathsf{Person}, \mathsf{Person} \rangle\rangle$ as follows:
$\mathsf{inverse\_inclusion\_merge}(\langle\langle \mathsf{hasSpouse}, \mathsf{Person}, \mathsf{Person} \rangle\rangle, \langle\langle \mathsf{spouse} \rangle\rangle)$
$\mathsf{redirect\_inclusion\_constraint}(\langle\langle \subseteq, \mathsf{Person}, \mathsf{spouse} \rangle\rangle, \langle\langle \mathsf{Person{:}id} \rangle\rangle)$
which would generate a new node $\langle\langle \mathsf{spouse} \rangle\rangle$ which has a mandatory constraint between it and the edge $\langle\langle \mathsf{hasSpouse}, \mathsf{Person}, \mathsf{Person} \rangle\rangle$ and is a subset of the node $\langle\langle \mathsf{Person} \rangle\rangle$. Applying this process to both ends of the edge: $\langle\langle \mathsf{hasSpouse}, \mathsf{Person}, \mathsf{Person} \rangle\rangle$:
$\mathsf{inverse\_inclusion\_merge}(\langle\langle \mathsf{hasParent}, \mathsf{Person\#1}, \mathsf{Person\#2} \rangle\rangle, \langle\langle \mathsf{child} \rangle\rangle)$
$\mathsf{redirect\_inclusion\_constraint}(\langle\langle \subseteq, \mathsf{Person}, \mathsf{child} \rangle\rangle, \langle\langle \mathsf{Person{:}id} \rangle\rangle)$
$\mathsf{inverse\_inclusion\_merge}(\langle\langle \mathsf{hasParent}, \mathsf{Person\#2}, \mathsf{Person\#1} \rangle\rangle, \langle\langle \mathsf{parent} \rangle\rangle)$
$\mathsf{redirect\_inclusion\_constraint}(\langle\langle \subseteq, \mathsf{Person}, \mathsf{parent} \rangle\rangle, \langle\langle \mathsf{Person{:}id} \rangle\rangle)$

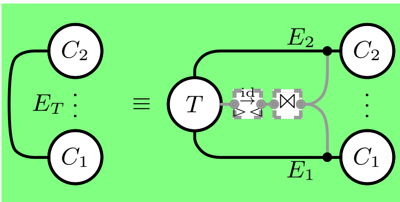
generates two new nodes node:$\langle\langle \mathsf{parent} \rangle\rangle$ and node:$\langle\langle \mathsf{child} \rangle\rangle$ each with a subset constraint with the node representing the key of person.

(C) Represent non-functional properties in OWL as separate tables in the relational model.

The HDM graph equivalence mapping: $\mathsf{identity\_edge\_merge}(\langle\langle \mathsf{T}, \mathsf{C_1}, \mathsf{C_2} \rangle\rangle), \langle\langle \mathsf{E_T} \rangle\rangle)$ that replaces the edge by a new node $\langle\langle \mathsf{T} \rangle\rangle$ connected to the two original nodes via two edges where there are $\lhd$, $\rhd$, and $\overset{\text{id}}{\to}$ constraints between the new node and the natural join of the two new edges.

Table 2: **HDM Representations for OWL 2 RL Property Axioms**

OWL 2 Construct	DL Syntax	HDM Representation
objectProperty	P	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle$
dataProperty	R	$\langle\langle \mathsf{rdf{:}datatype}\rangle\rangle$, $\langle\langle \mathsf{R}, \mathsf{C}, \mathsf{rdf{:}datatype}\rangle\rangle$
subPropertyOf	$\mathsf{P} \sqsubseteq \mathsf{Q}$	$\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{P}, \mathsf{Q}\rangle\rangle$
equivalentProperty	$\mathsf{P} \equiv \mathsf{Q}$	$\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{P}, \mathsf{Q}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \subseteq, \mathsf{Q}, \mathsf{P}\rangle\rangle$
propertyDisjointWith	$\mathsf{P} \sqcap \mathsf{Q} \sqsubseteq \bot$	$\mathsf{cons}{:}\langle\langle \not\cap, \mathsf{P}, \mathsf{Q}\rangle\rangle$
inverseOf	$\mathsf{P} \equiv \mathsf{Q}^{-}$	$\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle \mathsf{C}_2,\mathsf{C}_1\rangle}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle, \langle\langle \mathsf{Q}, \mathsf{C}_2, \mathsf{C}_1\rangle\rangle\rangle\rangle$ $\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle \mathsf{C}_1,\mathsf{C}_2\rangle}\langle\langle \mathsf{Q}, \mathsf{C}_2, \mathsf{C}_1\rangle\rangle, \langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle\rangle\rangle$
symmetricProperty	$\mathsf{P} \equiv \mathsf{P}^{-}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle$ $\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle \mathsf{D},\mathsf{C}\rangle}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle, \langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle\rangle\rangle$
transitiveProperty	$\mathsf{P} \circ \mathsf{P} \sqsubseteq \mathsf{P}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle$ $\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle \mathsf{P}\#1.\mathsf{C}_1,\mathsf{P}\#2.\mathsf{C}_2\rangle}\mathsf{P} \bowtie \mathsf{P}, \langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle\rangle\rangle$
propertyChain	$\mathsf{P1} \circ \ldots \circ \mathsf{Pn} \sqsubseteq \mathsf{P}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}_1, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle$, . . . , $\mathsf{edge}{:}\langle\langle \mathsf{P}_n, \mathsf{C}_n, \mathsf{C}_{n+1}\rangle\rangle$ $\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_{n+1}\rangle\rangle$ $\mathsf{cons}{:}\langle\langle \subseteq, \pi_{\langle \mathsf{C}_1,\mathsf{C}_{n+1}\rangle}\mathsf{P}_1 \bowtie \ldots \bowtie \mathsf{P}_n, \langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_{n+1}\rangle\rangle\rangle\rangle$
functionalProperty	$\top \sqsubseteq \leq 1\mathsf{P}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{C}_1, \langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle\rangle\rangle$
inverseFunctionalProperty	$\top \sqsubseteq \leq 1\mathsf{P}^{-}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{C}_2, \langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle\rangle\rangle$
irreflexiveProperty	$\top \sqsubseteq \neg\exists\ \mathsf{P.self}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}_1, \mathsf{C}_2\rangle\rangle$, $\mathsf{edge}{:}\langle\langle \mathsf{Q}, \mathsf{D}_1, \mathsf{D}_2\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \not\cap, \mathsf{P}, \mathsf{Q}\rangle\rangle$ $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{D}_1, \mathsf{Q}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \rhd, \mathsf{D}_1, \mathsf{Q}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \xrightarrow{\mathrm{id}}, \mathsf{D}_1, \mathsf{Q}\rangle\rangle$
selfRestriction	$\exists\ \mathsf{P.self}$	$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \rhd, \mathsf{C}, \mathsf{P}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{C}, \mathsf{P}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \xrightarrow{\mathrm{id}}, \mathsf{C}, \mathsf{P}\rangle\rangle$
key		$\mathsf{edge}{:}\langle\langle \mathsf{P}, \mathsf{C}, \mathsf{D}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \rhd, \mathsf{C}, \mathsf{P}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \lhd, \mathsf{C}, \mathsf{P}\rangle\rangle$, $\mathsf{cons}{:}\langle\langle \xrightarrow{\mathrm{id}}, \mathsf{C}, \mathsf{P}\rangle\rangle$



For example, applying the mapping to $\mathsf{edge}{:}\langle\langle \mathsf{hasParent}, \mathsf{child}, \mathsf{parent}\rangle\rangle$ is done by:
$\mathsf{inverse\_identity\_edge\_merge}(\langle\langle \mathsf{hasParent}, \mathsf{child}, \mathsf{parent}\rangle\rangle, \langle\langle \mathsf{hasParent}\rangle\rangle)$
and would generate HDM objects equivalent to the hasParent table in Section 3.

The result of steps (A), (B) and (C) is an HDM graph that is a forest of two-level trees, with subset constraints linking the leaf nodes.

6. BUILDING AN RDB SCHEMA

Step (5) in Figure 1 builds a core relational database schema from the HDM OWL2RL+rel schema. By core, we mean it defines tables, columns, and primary keys that allow data to be held, without defining any triggers or constraints that would affect the open-world or closed-world interpretation of the data. In outline, the HDM OWL2RL+rel schema is a forest of two level trees, where root nodes are connected to a number of leaf nodes via edges.

From our methodology in previous section, we can derive three production rules for building a relational schema of the general form *HDM patterns* $\rightsquigarrow$ *Relational construct*. For HDM nodes that are roots of the tree (and thus came from OWL 2 RL classes), we map them into a table of the same name:

Class: $\mathsf{node}{:}\langle\langle \mathsf{T}\rangle\rangle \rightsquigarrow \mathsf{table}{:}\langle\langle \mathsf{T}\rangle\rangle$

The leaf nodes that have been created by Steps (A) or (B) of the previous section will be mapped into columns:

Property: $\mathsf{edge}{:}\langle\langle \_, \mathsf{T}, \mathsf{C}_2\rangle\rangle, \mathsf{node}{:}\langle\langle \mathsf{C}_2\rangle\rangle, \mathsf{cons}{:}\langle\langle \rhd, \mathsf{C}, \langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle\rangle\rangle$
$\rightsquigarrow \mathsf{column}{:}\langle\langle \mathsf{T}, \mathsf{C}\rangle\rangle$

Finally, for those edges for which we have defined sufficient constraints to intrepret it as the key (*i.e.* resulting from Step (A) above) we can define a primary key of the table.

$\mathsf{edge}{:}\langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle, \mathsf{cons}{:}\langle\langle \rhd, \mathsf{C}, \langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle\rangle\rangle, \mathsf{cons}{:}\langle\langle \rhd, \mathsf{T}, \langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle\rangle\rangle,$
$\mathsf{cons}{:}\langle\langle \lhd, \mathsf{T}, \langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle\rangle\rangle, \mathsf{cons}{:}\langle\langle \xrightarrow{\mathrm{id}}, \mathsf{T}, \langle\langle \_, \mathsf{T}, \mathsf{C}\rangle\rangle\rangle\rangle$
$\rightsquigarrow \mathsf{primary\_key}{:}\langle\langle \mathsf{T}, \mathsf{C}\rangle\rangle$

These rules will result in a relational schema identical to the one shown in Example 3.1.

7. HANDLING CONSTRAINTS IN OWS

We now outline the process of handling the unconverted HDM constraints under OWS which corresponds to Step (6) in Figure 1. The approach we follow was inspired by the work of [15, 9] in which we first derive logical triggers over the relational schema that is resulted from the previous section and then implement those logical triggers as SQL physical triggers on a particular target DBMS as will be illustrated in Section 9. The logical triggers are translated from the HDM constructors according to the general form:

HDM construct $\rightsquigarrow$ **when** *event* **if** *condition* **then** *action*.

where *event* is the insertion process of a data value into a table. There are two types of event: if *event* is prefixed with $^{-}$ then *condition* and *action* are executed before the insertion, whilst if *event* is prefixed with $^{+}$ then *condition* and *action* are executed after the insertion. SQL before triggers (in pl/pgSQL) or instead of triggers (in Transact-SQL) are used to implement $^{-}$ events, and after triggers, used for $^{+}$ events. The *condition* is a Datalog query over the database, and *action* is either a data tuple to insert, *ignore* (ignoring this insertion that caused the trigger to execute) and *rollback* (rollback the transaction). The logical triggers can be translated into SQL physical triggers following the approach given in [15].

One basic rule deals with the notion that because of the open world nature of reasoning, we might repeatedly infer the same fact, and thus we have to prevent duplicate updates to a table. This is implemented by the logical triggers:

class: $\quad \mathsf{node}{:}\langle\!\langle \mathsf{C} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{-}C(x) \textbf{ if } C(x) \textbf{ then } \textit{ignore}$$

This means that when a value is inserted into a table, before the actual insert is done, a check is made to determine if the value is already present in the table, and if so, the insert is ignored.

The logical trigger for subClassOf generates a trigger that implies that each insertion to one class will automatically generate the same insertion(s) to its super class(es). Therefore, the consistency of the relations between classes is maintained.

subClassOf: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \mathsf{D} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}C(x) \textbf{ if } \mathsf{true} \textbf{ then } D(x)$$

A similar logical trigger for subPropertyOf is generated:

subPropertyOf: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{P}, \mathsf{Q} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}P(x,y) \textbf{ if } \mathsf{true} \textbf{ then } Q(x,y)$$

Thus, for rules (1) and (22) we can derive the following logical triggers:

$\textbf{when } {}^{+}\mathsf{Man}(x) \textbf{ then } \mathsf{Person}(x)$

$\textbf{when } {}^{+}\mathsf{hasParent}(x,y) \textbf{ then } \mathsf{hasGrandParent}(x,y)$

The logical trigger for complementOf generates a trigger which implies that individuals in class D should not appear in class C and vice versa.

complementOf: $\quad \mathsf{cons}{:}\langle\!\langle \not\cap, \mathsf{C}, \mathsf{D} \rangle\!\rangle$

$\qquad\qquad\quad \mathsf{cons}{:}\langle\!\langle \cup, \mathsf{Thing}, \mathsf{D}, \mathsf{C} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{-}C(x) \textbf{ if } D(x) \text{ then } \textit{rollback}$$
$$\rightsquigarrow \textbf{when } {}^{-}D(x) \textbf{ if } C(x) \text{ then } \textit{rollback}$$

Thus, for rule (3) we can derive the following logical triggers:

$\textbf{when } {}^{-}\mathsf{Man}(x) \textbf{ if } \mathsf{Woman}(x) \textbf{ then } \textit{rollback}$

$\textbf{when } {}^{-}\mathsf{Woman}(x) \textbf{ if } \mathsf{Man}(x) \textbf{ then } \textit{rollback}$

The logical trigger for equivalentClass generates a trigger which implies that each insertion to one class will automatically generate the same insertion(s) to the other class.

equivalentClass: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \mathsf{D} \rangle\!\rangle$

$\qquad\qquad\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{D}, \mathsf{C} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}C(x) \textbf{ if } \mathsf{true} \textbf{ then } D(x)$$
$$\rightsquigarrow \textbf{when } {}^{+}D(x) \textbf{ if } \mathsf{true} \textbf{ then } C(x)$$

The construct someValuesFrom ∃ P.D defines a set of individuals x that has atleast one tuple like (x,y) in P and y is in D. OWL 2 RL restricts the appearance of a someValuesFrom to be only in a subclass expression, so the logical trigger only contains the situation ∃ P.D ⊑ C:

someValuesFrom: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \exists\mathsf{P.D}, \mathsf{C} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}P(x,y) \textbf{ if } D(y) \textbf{ then } C(x)$$
$$\rightsquigarrow \textbf{when } {}^{+}D(y) \textbf{ if } P(x,y) \textbf{ then } C(x)$$

The logical trigger generates a trigger that checks for individuals that satisfy the existensial restriction and inserts them to table C. Thus, for rule (4) we can derive the following logical triggers:

$\textbf{when } {}^{+}\mathsf{hasChild}(x,y) \textbf{ if } \mathsf{Person}(y) \textbf{ then } \mathsf{Parent}(x)$

$\textbf{when } {}^{+}\mathsf{Person}(y) \textbf{ if } \mathsf{hasChild}(x,y) \textbf{ then } \mathsf{Parent}(x)$

The hasValue construct has two situations in which it may appear (subClass and a superClass) and consequently two logical triggers are generated respectively as follows:

hasValue: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \exists\mathsf{P.a}, \mathsf{C} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}P(x,a) \textbf{ if } \mathsf{true} \textbf{ then } C(x)$$

hasValue: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \exists\mathsf{P.a} \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } {}^{+}C(x) \textbf{ if } \mathsf{true} \textbf{ then } P(x,a)$$

If a certain class was a subclass of a hasValue expression, then we should insert (x,a) to the table P whenever there is an insertion of x to C. On the other hand, if the expression is a subset of a class, a tuple (x,a) inserted to P will invoke the trigger which will insert x to the table C. Thus, for rule (5) we can derive the following logical triggers:

$\textbf{when } {}^{+}\mathsf{hasParent}(x, John) \textbf{ if } \mathsf{true} \textbf{ then } \mathsf{JohnsChildren}(x)$

$\textbf{when } {}^{+}\mathsf{JohnsChildren}(x) \textbf{ if } \mathsf{true} \textbf{ then } \mathsf{hasChild}(x, John)$

The logical trigger for inverseOf keeps the relation of inverse properties which means if P_1 and P_2 are inverse properties and (x,y) is a property instance of property P_1, then (y,x) has to be an instance of property P_2.

inverseOf: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_2},\mathsf{C_1} \rangle} \langle\!\langle \mathsf{P_1}, \mathsf{C_1}, \mathsf{C_2} \rangle\!\rangle, \langle\!\langle \mathsf{P_2}, \mathsf{C_2}, \mathsf{C_1} \rangle\!\rangle \rangle\!\rangle,$

$\qquad\qquad \mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_1},\mathsf{C_2} \rangle} \langle\!\langle \mathsf{P_2}, \mathsf{C_2}, \mathsf{C_1} \rangle\!\rangle, \langle\!\langle \mathsf{P_1}, \mathsf{C_1}, \mathsf{C_2} \rangle\!\rangle \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } +P_1(x,y) \textbf{ if } \neg P_2(y,x) \textbf{ then } P_2(y,x)$$

In this case, a trigger in table P_1 checks each of its inverse tuples and inserts them to property table P_2 if the inverse tuples do not exist in P_2.

For a symmetricProperty, after inserting a tuple (x,y) to property P, the symmetric tuple (y,x) will be inserted to P by a trigger.

symmetricProperty: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_2},\mathsf{C_1} \rangle} \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2} \rangle\!\rangle, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2} \rangle\!\rangle \rangle\!\rangle$

$$\rightsquigarrow \textbf{when } +P(x,y) \textbf{ if } \mathsf{true} \textbf{ then } P(y,x)$$

For instance, for rule (26) we can derive the following logical trigger:

$\textbf{when } {}^{+}\mathsf{hasSpouse}(x,y) \textbf{ then } \mathsf{hasSpouse}(y,x)$

For a transitiveProperty, after inserting a tuple (x,y), it will try to find if tuple (y,z) exists. If so, it will then insert tuple (x,z). Similarly, it will try to find if tuple (z,x) exists and then insert tuple (z,y).

transitiveProperty: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{P\#1.C_1},\mathsf{P\#2.C_2} \rangle} \mathsf{P} \bowtie \mathsf{P}, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2} \rangle\!\rangle \rangle\!\rangle$

$$\begin{aligned} \rightsquigarrow \textbf{when } +P(x,y) \quad & \textbf{if } P(y,z) \textbf{ then } P(x,z) \\ & \textbf{if } P(z,x) \textbf{ then } P(z,y) \end{aligned}$$

For example, for rule (30) we can derive the following logical trigger:

$$\begin{aligned} \textbf{when } {}^{+}\mathsf{hasAncestor}(x,y) \quad & \textbf{if } \mathsf{hasAncestor}(y,z) \\ & \textbf{then } \mathsf{hasAncestor}(x,z) \\ & \textbf{if } \mathsf{hasAncestor}(z,x) \\ & \textbf{then } \mathsf{hasAncestor}(z,y) \end{aligned}$$

propertyChain allows for a property to be defined from the concatenation of two or more other properties. The logical triggers are as follows:

propertyChain: $\quad \mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_1},\mathsf{C_{n+1}} \rangle} \mathsf{P_1} \bowtie \ldots \bowtie \mathsf{P_n}, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_{n+1}} \rangle\!\rangle \rangle\!\rangle$

$$\begin{aligned} \rightsquigarrow \textbf{when } {}^{+}P_1(x,y) \quad & \textbf{if } P'_{2,n}(y,z) \textbf{ then } P(x,z) \\ \rightsquigarrow \textbf{when } {}^{+}P_n(y,z) \quad & \textbf{if } P'_{1,n-1}(x,y) \textbf{ then } P(x,z) \\ \rightsquigarrow \textbf{when } {}^{+}P_i(p,q) \quad & \textbf{if } P'_{1,i-1}(x,p), P'_{i+1,n}(q,z) \\ & \textbf{then } P(x,z) \;{}_{1<i<n} \end{aligned}$$

$$*P'_{m,n}(x,y) = {}^{\pi_{P_m.domain,P_n.range}}_{\sigma_{P_j.range=P_{j+1}.domain}} (P_m \times \ldots \times P_n)_{m \le j < n}$$

The first and second situations mean that if there is an insertion to the first or last subchain, the trigger will treat the remaining subchains as a join unit and search data matched inside the unit. The third scenario handles the insertion to the middle subchains by creating two join units and then fetch matching data from them. For example, for rule (25) we can derive the following logical triggers:

$$\begin{aligned} \textbf{when } {}^{+}\mathsf{hasParent\#1}(x,y) \quad & \textbf{if } \mathsf{hasParent\#2}(y,z) \\ & \textbf{then } \mathsf{hasGrandParent}(x,z) \\ \textbf{when } {}^{+}\mathsf{hasParent\#2}(y,z) \quad & \textbf{if } \mathsf{hasParent\#1}(x,y) \\ & \textbf{then } \mathsf{hasGrandParent}(x,z) \end{aligned}$$

8. HANDLING CONSTRAINTS IN CWS

In this Section, we show an alternative way of handling the unconverted HDM constraints under CWS which corresponds to Step 7 in Figure 1.

The basic idea for handling constraints in CSW is we create SQL constraints to check each HDM constraints after data is inserted. We follow the same approach we have used in OWS that, we first generate logical constraints and then show their physical implementation (SQL physical constraints) in Section 9. Logical constraints are translated from the HDM constructors according to productions rules of the general form:

$$\textit{HDM construct} \rightsquigarrow \textbf{when } \textit{event} \textbf{ if } \textit{condition} \textbf{ then } \textit{action}.$$

which is similar to the general form of logical triggers in OWS. The *event* is always happened after the data insertion (denoted by $^+$), since SQL constraints cannot be verified with no data inserted. The *condition* is logical check queries derived from an HDM constraint and the *action* is automatically performed by SQL Server either to *allow* or to *rollback* the insertions. Next, we demonstrate certain logical constraints for handling constraints in CWS.

The logical constraint for subClassOf will verify that if the data inserted to one class also in its super class(es).

subClassOf: $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \mathsf{D}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+C(x)$ **if** $\neg D(x)$ **then** *rollback*

Similarly, the logical constraint for subPropertyOf is to verify all tuples inserted to a property exist in its super properties:

subPropertyOf: $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{P}, \mathsf{Q}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+P(x,y)$ **if** $\neg Q(x,y)$ **then** *rollback*

For example, for rules (1) and (22) we can derive the following logical constraints:

when $^+\mathsf{Man}(x)$ **if** $\neg\mathsf{Person}(x)$ **then** *rollback*

when $^+\mathsf{hasParent}(x,y)$ **if** $\neg\mathsf{hasGrandParent}(x,y)$ **then** *rollback*

The logical constraints for complementOf (e.g. C ≡ ¬ D) which are shown below only allow to insert data to the C if the data is not in the table D, and vice versa:

complementOf: $\mathsf{cons}{:}\langle\!\langle \not\cap, \mathsf{C}, \mathsf{D}\rangle\!\rangle$

$\mathsf{cons}{:}\langle\!\langle \cup, \mathsf{Thing}, \mathsf{D}, \mathsf{C}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+C(x)$ **if** $D(x)$ then *rollback*

$\rightsquigarrow$ **when** $^+D(x)$ **if** $C(x)$ then *rollback*

Thus, for rule (3) we can derive the following logical constraints:

when $^-\mathsf{Man}(x)$ **if** $\mathsf{Woman}(x)$ **then** *rollback*

when $^-\mathsf{Woman}(x)$ **if** $\mathsf{Man}(x)$ **then** *rollback*

The logical constraint for equivalentClass generates a check that verifies that if data is inserted to one table, it is also in the equivalent table of this class.

equivalentClass: $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \mathsf{D}\rangle\!\rangle$

$\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{D}, \mathsf{C}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+C(x)$ **if** $\neg D(x)$ **then** *rollback*

$\rightsquigarrow$ **when** $^+D(x)$ **if** $\neg C(x)$ **then** *rollback*

The expression of someValuesFrom (∃P.D) only appears in a subclass expression in OWL 2 RL, and a constraint check should be generated to verify x is in the table C, when there are an insertion of (x,y) to the table P and another insertion of y to the table D, of which the logical constraint is shown below:

someValuesFrom: $\mathsf{cons}{:}\langle\!\langle \subseteq, \exists\mathsf{P.D}, \mathsf{C}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+P(x,y), D(y)$ **if** $\neg C(x)$ **then** *rollback*

For example, for rule (4) we can derive the following logical constraint to check that whether the individuals that satisfy the existential restriction also exist in the table Parent:

when $^+\mathsf{hasChild}(x,y), \mathsf{Person}(y)$ **if** $\neg\mathsf{Parent}(x)$ **then** *rollback*

The hasValue construct has two situations in which it may appear and consequently two logical constraints are generated respectively as follows:

hasValue: $\mathsf{cons}{:}\langle\!\langle \subseteq, \exists\mathsf{P.a}, \mathsf{C}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+P(x,a)$ **if** $\neg C(x)$ **then** *rollback*

hasValue: $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{C}, \exists\mathsf{P.a}\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+C(x)$ **if** $\neg P(x,a)$ **then** *rollback*

In the first case, the logical constraint checks if a tuple (x,a) is in the table P, then x should also appear in the table C and vice versa for the second case. Thus, for rule (5) we can derive the following logical constraints:

when $^+\mathsf{hasParent}(x, John)$ **if** $\neg\mathsf{JohnsChildren}(x)$ **then** *rollback*

when $^+\mathsf{JohnsChildren}(x)$ **if** $\neg\mathsf{hasParent}(x, John)$ **then** *rollback*

The logical constraint for inverseOf checks that if (x,y) is a property instance of property P_1 then (y,x) has to be an instance of property P_2.

inverseOf: $\mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_2},\mathsf{C_1}\rangle}\langle\!\langle \mathsf{P_1}, \mathsf{C_1}, \mathsf{C_2}\rangle\!\rangle, \langle\!\langle \mathsf{P_2}, \mathsf{C_2}, \mathsf{C_1}\rangle\!\rangle\rangle\!\rangle$,

$\mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_1},\mathsf{C_2}\rangle}\langle\!\langle \mathsf{P_2}, \mathsf{C_2}, \mathsf{C_1}\rangle\!\rangle, \langle\!\langle \mathsf{P_1}, \mathsf{C_1}, \mathsf{C_2}\rangle\!\rangle\rangle\!\rangle$

$\rightsquigarrow$ **when** $+P_1(x,y)$ **if** $\neg P_2(y,x)$ **then** *rollback*

For a symmetricProperty, we create a similar check which verifies that if a tuple (x,y) is inserted into the table P, its symmetric tuple (y,x) is also in P.

symmetricProperty: $\mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_2},\mathsf{C_1}\rangle}\langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2}\rangle\!\rangle, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2}\rangle\!\rangle\rangle\!\rangle$

$\rightsquigarrow$ **when** $+P(x,y)$ **if** $\neg P(y,x)$ **then** *rollback*

For instance, for rule (26) we can derive the following logical constraint:

when $^+\mathsf{hasSpouse}(x,y)$ **if** $\neg\mathsf{hasSpouse}(y,x)$ **then** *rollback*

For a transitiveProperty, a check will verify a tuple (x,z) is in the table P, if there are tuples (x,y) and (y,z) in P:

transitiveProperty: $\mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{P\#1.C_1},\mathsf{P\#2.C_2}\rangle}\mathsf{P} \bowtie \mathsf{P}, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_2}\rangle\!\rangle\rangle\!\rangle$

$\rightsquigarrow$ **when**$^+P(x,y), P(y,z)$ **if** $\neg P(x,z)$ **then** *rollback*

For example, for rule (30) we can derive the following logical rule:

when $^+\mathsf{hasAncestor}(x,y), \mathsf{hasAncestor}(y,z)$
if $\neg\mathsf{hasAncestor}(x,z)$ **then** *rollback*

For a propertyChain, we generate the following logical constraint:

propertyChain: $\mathsf{cons}{:}\langle\!\langle \subseteq, \pi_{\langle \mathsf{C_1},\mathsf{C_{n+1}}\rangle}\mathsf{P_1} \bowtie \ldots \bowtie \mathsf{P_n}, \langle\!\langle \mathsf{P}, \mathsf{C_1}, \mathsf{C_{n+1}}\rangle\!\rangle\rangle\!\rangle$

$\rightsquigarrow$ **when** $^+P_1(x,y), P'_{2,n}(y,z)$ **if** $\neg P(x,z)$ **then** *rollback*

$*P'_{m,n}(x,y) = {}^{\pi_{P_m.domain, P_n.range}}_{\sigma_{P_j.range=P_{j+1}.domain}}(P_m \times \ldots \times P_n)_{m \le j < n}$

For example, for rule (25) we can derive the following logical rule:

when $^+\mathsf{hasParent\#1}(x,y), \mathsf{hasParent\#2}(y,z)$
if $\neg\mathsf{hasGrandParent}(x,z)$ **then** *rollback*

9. IMPLEMENTATION OF OWS & CWS

After generating logical triggers and constraints, SQL physical triggers and constraints can be implemented intuitively by SQL statements. Physical trigger generation is based on trigger translation rules introduced in [15].

The implementation from logical constraints to physical constraints is slightly different. If a logical constraint is to check the subsumption relationships between the value of columns (such as subClassOf, subPropertyOf, symmetricProperty and inverseOf), foreign keys are used to implement this logical check. For example, For example, the HDM subclass constraint $\mathsf{cons}{:}\langle\!\langle \subseteq, \mathsf{Man}, \mathsf{Person}\rangle\!\rangle$ representing rule (1) can be achieved using a FK, FK_Man_isa_Person, between Man and Person. Furthermore, HDM constraints resembling complementOf, someValuesFrom, transitiveProperty, and propertyChain can be implemented in a CWS approach by writing functions that check if the constraint holds. Next, we show examples of physical triggers and constraints translated from several HDM constructs.

The first example could be a subsumption relationship between properties, such as the rule (23). In the OWS implementation, an after trigger called hasGrandParent_subOf_hasAncestor shown in Figure 5(a) will be created for the table hasGrandParent and it will insert to the table hasAncestor the data inserted to hasGrandParent. However, in a CWS implementation, we can use a constraint of foreign key called FK_hasGrandParent_subOf_hasAncestor from the two columns of hasGrandParent to columns of the table hasAncestor (shown in Figure 5(b)).

Figure 4: Triggers and Constraints for subPropertyOf.

```
CREATE TRIGGER hasGrandParent_subOf_hasAncestor
ON hasGrandParent
AFTER INSERT AS BEGIN
        INSERT INTO hasAncestor
        SELECT grandchild, grandparent
        FROM inserted
        EXCEPT SELECT descendant, ancestor
        FROM hasAncestor
END
```

(a) SQL Trigger for subPropertyOf.

```
ALTER TABLE hasGrandParent
ADD CONSTRAINT FK_hasGrandParent_isa_hasAncestor
FOREIGN KEY (grandchild, grandparent)
REFERENCES hasAncestor(descendant, ancestor)
ALTER TABLE Man
NOCHECK CONSTRAINT FK_hasGrandParent_isa_hasAncestor
```

(b) SQL Constraint for subPropertyOf.

Figure 5: Triggers and Constraints for propertyChain.

```
CREATE TRIGGER hasParent_chain_hasGrandParent
ON hasParent
AFTER INSERT AS BEGIN
        INSERT INTO hasGrandParent
        SELECT p1.child, p2.parent
        FROM hasParent AS p1 JOIN hasParent AS p2
        ON p1.parent = p2.child
        EXCEPT SELECT * FROM hasGrandParent
END
```

(a) SQL Trigger for propertyChain.

```
CREATE FUNCTION dbo.checkPropertyChain()
RETURNS BIT AS BEGIN
    DECLARE @Exists BIT
    IF NOT EXISTS (
            SELECT p1.child, p2.parent
            FROM hasParent AS p1 JOIN hasParent AS p2
            ON p1.parent = p2.child
            EXCEPT SELECT grandchild, grandparent
            FROM hasGrandParent)
    BEGIN SET @Exists = 1 END
    ELSE BEGIN SET @Exists = 0 END
    RETURN @Exists
END
ALTER TABLE hasParent
ADD CONSTRAINT CK_propertyChain
CHECK (dbo.checkPropertyChain() = 1)
```

(b) SQL Constraint for propertyChain.

Note that, since the physical constraint check will be performed after data insertions, so we will disable the constraint check when loading the data and enable it after the update transaction is committed.

Another example could be a new feature of OWL 2 RL which is propertyChain, exemplified by the rule (25). In OWS, we create a trigger to insert the self join values of the table hasParent to the table hasGrandParent. However, the physical constraint is more complex which cannot be implemented by a foreign key. Therefore we create a checking function which verifies the self-joint tuples of the table hasParent (i.e. $\pi_{\text{hasParent\#1Domain},\text{hasParent\#2Range}}\text{hasParent\#2} \bowtie \text{hasParent\#1}$) are in the table hasGrandParent. The trigger and constraint are shown in Figure 6(a) and Figure 6(b), respectively (Note that in the physical constraint we use the BIT value 1 to denote TRUE).

10. EVALUATION OF OWLREL

In this section, we show the evaluation of our system with regards to OWS and CWS. For evaluating OWLRel under OWS, we considered the completeness, efficiency, and scalability metrics. For evaluating OWLRel's completeness, we have run the 14 queries of the well known **Lehigh University Ontology Benchmark (LUBM)** [6] with original datasets generated from LUBM's A-Box generator, and also more exhaustive A-Boxes generated by SyGENiA [22]. For evaluating the efficiency and scalability, we have run different sizes of LUBM and checked the scalability of the system in terms of data loading and query processing, and compared query processing time with another semantic reasoner, OWLIM-Lite [8]. Under a CWS setting, we ran some checks to guarantee the soundness of our results. In Table 3, we show the time required for processing each step of OWLRel with a total time of 3.40 (min) in OWS and of 3.39 (min) in CWS.

Table 3: OWLRel Performance Report of LUBM.

OWLRel Steps	Time (s)
1 & 2 Loading and Classification Time:	10.48
3 HDM Transformation Time:	34.48
4 HDM InterModel Transformation Time:	120.92
5 Relational Transformation Time:	37.00
6 OWS Transformation Time:	0.84
7 CWS Transformation Time:	0.29

OWLRel and OWLIM-Lite were tested on a machine with 2 Intel Xeon E5345 with 2.33GHz CPUs and 8GB of memory, which runs a Microsoft SQL Server 2005 database.

10.1 Evaluation Data

LUBM. The LUBM ontology describes concepts in a university domain. It comprises a T-Box which contains several OWL classes, properties and a number of OWL features such as, subClassOf, subPropertyOf, inverseOf, someValuesFrom, intersectionOf and transitiveProperty. Although the T-Box, is quite simple, LUBM contains a number of features that are beyond those permitted by the OWL 2 RL profile. Apart from the T-Box, LUBM contains 14 queries which we number L1-L14 and an A-Box generator to produce A-Boxes with different sizes. In our experiment, we use LUBM(n) to denote the LUBM A-Boxes of n universities. Each university contains about 100,000 individuals and property tuples.

SyGENiA. Only testing the original datasets of the LUBM benchmark would be limited in terms of completeness, since LUBM's original A-Boxes are not general and exhaustive enough. SyGENiA is able to generate a more complex A-Box for a given query and a T-Box. Thus, we further evaluate our system using another 14 A-Boxes generated by SyGENiA for the 14 queries of LUBM. For each A-Box, we set the number of assertions to be 1000.

10.2 Evaluation of OWLRel under OWS

10.2.1 Completeness of OWLRel

OWLRel shows a 100% completeness level over LUBM for both, the original A-Boxes and the more exhaustive A-Boxes generated by SyGENiA, which means that our system is the better than OWLIM, Minerva [27], HAWK [17] and Sesame [5] mentioned in [22]. Table 4 shows the completeness level of each query processing compared with OWLIM-Lite (The completeness results of OWLIM and Minerva are from [22]). As can be seen, OWLIM cannot process L6, L8 and L10 completely. One reason for the more complete query processing of OWLRel than OWLIM is that we are able to handle the existential qualification, which is not completely supported by OWLIM. Moreover, Minerva's completeness level is not that high, since it failed to provide complete answers for L5, L6, L7, L8, L10, L12 and L13.

Table 4: Completeness level over SyGENiA LUBM of OWLRel, OWLIM and Minerva.

System	L5	L6	L7	L8	L10	L12	L13
OWLRel	1	1	1	1	1	1	1
OWLIM	1	0.96	1	0.93	0.96	1	1
Minerva	0.89	0.87	0.90	0.76	0.87	0.66	0.24

10.2.2 Efficiency and Scalability of OWLRel

In order to test the scalability of our system and the efficiency of our data loading and query processing, we compared the execution time for answering each query of LUBM and compare it to OWLIM-Lite.

Query Processing. As can be seen from Table 5, we compared our system with OWLIM-Lite in terms of the execution time for each query. The results show that on average OWLRel was faster than OWLIM per query over all different sizes in our experiment. Moreover, the average query processing time of our system was roughly increased linearly when we doubled the size of A-Boxes, which means that OWLRel scaled over the experiment data.

Data Loading. The data loading time of OWLRel for LUBM(5), LUBM(10), LUBM(20) and LUBM(40) is 15(min), 55(min), 97(min) and 187(min), respectively. OWLRel performed quite fast data loading; for example, it was able to insert approximate 356.5 tuples into the database per second for LUBM(40). Moreover, the data loading time also was increased almost linearly when the data size was doubled, which means that OWLRel was also scalable for loading the A-Boxes.

10.3 Evaluation of OWLRel under CWS

Since there is not a good benchmark for evaluating our system in CWS, we just manually verified our constraint checking. For example, considering the rule (3), inserting John to table Woman has been rolled back, since John was already in table Man. On the other hand, loading the completely reasoned data has not generated any violations.

11. RELATED WORK

In the context of mapping ontological models to the relational model, most proposals (e.g. [24, 2]) suffer from one of these limitations: ignore OWL restrictions that do not have correspondences in the relational model, store the ontology in a fixed schema (adopting a meta-schema approach), not support OWL 2 ontologies, or do not adhere to the OWS characteristics of ontologies.

In the context of reasoning in an RDBMS, RDBMSs are not only capable of processing ontologies with large number of individuals, but also provides many benefits, such as transaction management, security, integrity control, and scalability [1].

McBrien et al. [16], classified the existing methods that support consistency checking over relational data into three types: Application-based reasoners (ABR), VBR, and TBR. VBR systems such as, DLDB2 uses SQL views to achieve type inference, while SQOWL and its extension SQOWL2 [9] are TBR systems like OWLRel that applies SQL triggers to infer new knowledge. ABR systems such as, SOR [10] and OWLIM rely on reasoners to perform type inference outside an RDBMS.

SOR (previously called Minerva) uses a standard tableuax-based DL reasoner to perform the classification of the T-Box. Subsequently, it generates rules to perform type inference outside the database then materialises the results of inferences inside the RDBMS which makes query processing fast. Since the type inference process, however, was implemented outside the database, SOR does not support transactional reasoning nor incremental reasoning in an RDBMS as opposed to OWLRel or SQOWL2. Moreover, the current version of SOR only supports OWL 1 DL.

OWLIM-Lite, a sub system of OWLIM, does its reasoning in memory. It performs materialisation while loading the A-Box just like OWLRel and SQOWL2, however, it can not handle existential quantifications which makes OWLRel and SQOWL2 more complete.

DLDB2 and DBOWL store their rules inside the database as views and do not materialise the infered closure at loading time. This results in very fast loading time, but slow query processing. On the other hand, SQOWL and more recently SQOWL2 compile T-Box rules into DBMS before and after trigger statements providing a forward chaining materialisation approach.

12. CONCLUSIONS & FUTURE WORK

The mapping of ontologies to relational models has been an active area of research during the past decade. In this paper, we have given a complete, lossless transformation of an OWL 2 RL ontology to a relational schema via a HDM under two approaches; OWS using SQL triggers, and CWS using SQL constraints. So far, OWLRel provides faster query processing time than OWLIM with respect to the LUBM benchmark and shows promising scalability results. Future works will be directed towards first, using other benchmarks like UBOM [11] as well as real-world data for conducting exhaustive experiments to assure the quality of our method and improving the scalability of our system to handle billions of assertions. Finally, we will consider performing the

Table 5: LUBM Query Processing time (ms) of OWLRel and OWLIM.

Size	System	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	Average
LUBM(5)	OWLRel	42	62	21	2	16	6	71	162	253	42	2	7	0	4	49.29
	OWLIM	119	70	1	5	10	98	3	166	222	1	1	8	2	13	51.36
LUBM(10)	OWLRel	79	121	42	4	31	11	22	215	175	81	3	12	0	9	57.5
	OWLIM	118	115	1	6	9	124	3	215	423	1	2	13	1	26	75.5
LUBM(20)	OWLRel	29	376	87	6	69	24	44	61	901	28	7	11	1	17	118.64
	OWLIM	161	172	1	5	6	106	3	463	911	1	2	24	1	56	136.57
LUBM(40)	OWLRel	56	216	33	11	20	10	84	125	2391	55	12	23	2	35	219.5
	OWLIM	246	450	1	5	7	199	2	847	1895	1	1	47	1	118	272.86

mappings as bidirectional i.e., from relational databases with triggers or constraints to an ontology.

13. REFERENCES

[1] I. Astrova, N. Korda, and A. Kalja. Storing OWL Ontologies in SQL Relational Databases. *IJECSE*, 1(4):242–247, 2007.

[2] P. Atzeni, P. Del Nostro, and S. Paolozzi. Ontologies and Databases: Going Back and Forth. In *Proceedings of the 4th ODBIS*. Citeseer, 2008.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. CUP, 2003.

[4] M. Boyd and P. McBrien. Comparing and Transforming between Data Models via an Intermediate Hypergraph Data Model. *Journal on Data Semantics*, IV:69–109, 2005.

[5] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *The Semantic Web–ISWC 2002*, pages 54–68. Springer, 2002.

[6] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the WWW*, 3(2):158–182, 2005.

[7] M. Horridge and S. Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *OWLED*, volume 529, pages 11–21, 2009.

[8] A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM–a pragmatic semantic repository for OWL. In *WISE 2005 Workshops*, pages 182–192. Springer, 2005.

[9] Y. Liu and P. McBrien. SQOWL2: Transactional Type Inference for OWL 2 DL in an RDBMS. In *Description Logics*, pages 779–790, 2013.

[10] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. SOR: a practical system for ontology storage, reasoning and search. In *Proceedings of the 33rd VLDB*, pages 1402–1405, 2007.

[11] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a Complete OWL Ontology Benchmark. In *ESWC*, pages 125–139, 2006.

[12] P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications a schema transformation approach. In *Proceedings of ER*, 1999.

[13] P. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01*, volume 2068 of *LNCS*, pages 330–345. Springer, 2001.

[14] P. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.

[15] P. McBrien, N. Rizopoulos, and A. Smith. SQOWL: Type Inference in an RDBMS. *ER*, pages 362–376, 2010.

[16] P. McBrien, N. Rizopoulos, and A. Smith. Type inference methods and performance for data in an RDBMS. In *Proceedings of the 4th International Workshop on SWIM*, SWIM '12, pages 6:1–6:8, New York, NY, USA, 2012. ACM.

[17] Z. Pan. HAWK: OWL Repository and Toolkit. Lehigh University, Bethlehem, 2008.

[18] Z. Pan, X. Zhang, and J. Heflin. DLDB2: A scalable multi-perspective semantic web repository. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 489–495. IEEE, 2008.

[19] A. Poulovassilis and P. McBrien. A General Formal Framework for Schema Transformation. *DKE*, 28(1):47–71, 1998.

[20] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[21] A. Smith, N. Rizopoulos, and P. McBrien. Automed Model Management. *ER*, pages 542–543, 2008.

[22] G. Stoilos, B. C. Grau, and I. Horrocks. How Incomplete is Your Semantic Web Reasoner? In *AAAI*, 2010.

[23] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297. Springer, 2006.

[24] E. Vysniauskas, L. Nemuraite, R. Butleris, and B. Paradauskas. Reversible Lossless Transformation From OWL 2 Ontologies into Relational Databases. *IJITCA*, 40(4):293–306, 2011.

[25] W3C. Web Ontology Language Guide, February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[26] W3C. OWL 2 Web Ontology Language New Features and Rationale, June 2009. http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/.

[27] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In *The Semantic Web–ASWC 2006*, pages 429–443. Springer, 2006.