Functional Dependencies in OWL A-BOX

Jean-Paul Calbimonte, Fabio Porto

École Polytechnique Fédérale de Lausanne - EPFL
Database Laboratory - Switzerland
{jean-paul.calbimonte, fabio.porto}@epfl.ch

Abstract. Functional Dependency (FD) has been extensively studied in database theory. Most recently there have been some works investigating the implications of extending Description Logics with functional dependencies. In particular the OWL ontology language offers the functional property property allowing simple functional dependency to be specified. As it turns out, more complex FD specified as concept constructors has been proved to lead to undecidability in the general case, which restricts its usage as part of TBOX. This paper departs from previous ones by restricting FDs applicability to instances in the ABOX. We specify FD as a new constructor, an OWL concept. FD instances are mapped to Horn clauses and evaluated against the ABOX according to user's desired behavior. The latter allows users to determine whether FDs should be interpreted as constraints, assertions or views. Our approach gives ontology users data guarantees usually found in databases, integrated with the ontology conceptual model.

Keywords: Functional Dependency, OWL, DL, Database, ABOX

1. Introduction

Data dependencies have been introduced as a general formalism for a large class of database constraints that augments the expressivity of database models [1]. Functional dependencies (FD) compose a particularly interesting data dependency [2] that elegantly model the relationships between attributes of a relation, used for defining primary keys and in normalization theory avoiding redundant data representation. Other important application of FD in database includes query rewriting [3] and query evaluation [4].

The semantics expressed through functional dependencies are equally relevant when specifying a conceptual schema by means of ontologies. As a matter of fact, it has been observed [5] that in data-centric applications users expect ontologies to offer mechanisms similar to

those found in the database area that guarantee the correctness of entered data. In particular, FDs allow users to explicitly state high-level constraints that once enforced can validate the current state of a Description Logics ABOX.

Indeed, in recent years, a bulk of prior research has investigated the implications of adding functional dependencies to ontology languages (eg. 6,9,11,12,17,20,21). These initiatives took one of two paths: extending a DL dialect with new FD concept constructor or adding FD (and key) as number constraints over concepts and relationships. It turns out that extending DL with a new FD concept construct requires re-evaluating the logical implication algorithms, which in the general case has been shown to lead to undecidability [6].

Thus, in this scenario, adding database like constraints to ontologies with decidability guarantees requires a DL dialect developer to constrain the expressivity of the adopted FD language.

Note, however, that such a limitation appears when one tries to extend the typing system in a TBOX. For many data-centric applications it is the case that the correctness of entered data maybe more relevant than the expressiveness of its type system. This work takes the latter assumption and proposes an extension of DL ontologies with database like expressive FD rules. Indeed, FDs are specified as instances of an application level ontological concept. The adopted FD language allows formulating complex FD rules, including multiple paths in both the antecedent and consequent of the rule (i.e. the latter is in fact transformed into multiple single consequent path clauses). Three types of FDs were considered: classical, keys and explicit dependencies. The first two correspond to typical database functional dependency whereas the last one is a particular case of tuple generating dependency [7].

We implemented ABOX FDs by mapping FD instances to Horn clauses [18,19] using the SWRL rule language [8]. Surprisingly, the effect of running the FD rules over the ABOX may achieve different results depending on the desired behavior. Three of such behaviors have been identified leading to the extension of traditional DL knowledge base. Firstly the constraint behavior indicates instances that do not comply with the FD rules. A second approach refines the unique name assumption in DL by identifying *sameAs* instances represented by different nominals and adding the corresponding axioms to the ABOX. Finally, a *view* behavior returns query results matching the FD specification.

The remaining of this paper is structured as follows. Section 2 discusses related work. Next, section 3 gives some necessary background and motivates the problem through examples. Section 4 presents a formal framework for the FD construct and discusses enforcement interpretation. Section 5 introduces the FD construct in OWL and section 6 presents a first prototype implementation. Finally, section 7 concludes.

2. Related Work

Functional dependencies have been extensively studied in databases as a formalism to extend database schema semantics [1,18]. In the field of Description Logics (DL), FDs have also been the subject of recent investigations.

In [9], Borgida and Weddell expressed the necessity of adding uniqueness constraints to semantic data models, specifically DL. They used CLASSIC [10] as target knowledge representation system for introducing a new FD constructor, similar in syntax to object-oriented database keys and slightly modified to represent classic FDs. As expected, this simple FD declaration does not affect the tractability of the sub-sumption algorithm.

A more general FD concept constructor for DL was later introduced by Khizder, Toman and Weddell in [11]. Their approach mainly focused on uniqueness constraints with the extension of *paths* to express role composition in FD declaration elements. The resulting DL is named $\mathcal{DLFD}$ and a translation from DL-Class to DLFD is proposed. The authors explored the complexity of logical implication problems in $\mathcal{DLFD}$, by proving equivalence with query answering in $\text{Datalog}_{\text{nS}}$ with some restrictions, leading to a polynomial time query evaluation.

Calvanese, De Giacomo and Lenzerini, interested in modeling database schemas as DL knowledge bases, proposed in [12] identification and FD assertions for the $\mathcal{DLR}$ language. The $\mathcal{DLR}$ language has the particularity of having features such as n-ary relationships, adequate for modeling database schemas. FDs and uniqueness constraints are mapped to $\mathcal{DLR}$ number restrictions and showed that reasoning with these fd assertions is EXPTIME decidable. Another interesting feature of $\mathcal{DLR}_{\mathit{ifd}}$ is its ability for representing Object-Oriented class operations (methods) using the fd construct [15]. Given $f(P_1,...,P_m):R$ an operation of a class C, with m parameters, each one belonging to classes $P_1,...,P_m$ respectively and with the result of f belonging to class R; the following FD assertion can be specified: $(\text{fd } f\ P_1,...,P_m\ 1,...,m+1 \rightarrow m+2)$.

Lutz et al. [20] first considered the case of adding keys to more expressive DLs. The result is the addition of a set of key definition statements in a so-called key box. Lutz et al. proved that these key constraints have an important impact on decidability. For instance satisfiability of concepts becomes undecidable in the general case. Decidability is NEXPTIME-complete if key boxes are restricted to a particular kind called *boolean key boxes*. Lutz and Milicic further explored the possibility of adding not only keys but also FDs to DLs with concrete domains in [21]. Although it would initially seem that FDs are weaker than uniqueness keys, their work showed that the impact on decidability and complexity of reasoning is equally dramatic in the language they defined, $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$.

Efforts like [17], [20], [21] added a new kind of constraint terms for FDs. On the other hand [11] introduced FDs as a new kind of concept constructor as we have seen. In [24] Toman and Weddell extend their previous efforts [6] by adding the possibility of using the FD concept constructor not only in top level and in the right hand side of inclusion dependencies ($\subseteq$). However this extension in the general case is shown to lead to undecidability. They regain decidability by focusing on a reduced DL where Path FDs occur only at top level or in monotone concept constructors.

One can observe that there is a clear compromise between expressivity of FDs and the decidability of the resulting DL dialect. Our approach departs from these problems by introducing FD as an application level construct, without changes to the typing system.

Motik and colleagues [5] elegantly discuss the role of constraints in ontologies and trace an interesting comparison to constraints in databases.

Finally, Ludascher et al [22] introduced a distinguishable witness predicate for holding instances not conforming to specified constraints. The approach we adopted for integrating constraints enforcement to our framework was inspired by the aforementioned proposal.

3. Preliminaries

In database theory, FDs have been seen as one of the most important concepts of relational modeling. It allows specifying dependencies between attributes of relations and provides the basis for normalization theory and relational keys. A FD is denoted as $X \rightarrow Y$, with X and Y being sets of attributes of a relation R. Such FD states that the values of the attributes in Y are uniquely defined by the attributes in X.

When transposing similar rules to the ontology world we discover that FDs could indeed be very useful to enrich the expressivity of the knowledge representation. Take for instance a flight ontology. We can partially model the Flight, Airport and Gate concepts and their linking roles, as shown in Figure 1:

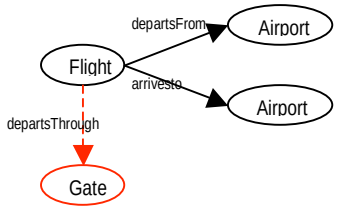

Fig. 1. FD for a flight and gate

In this representation the departsFrom and arrivesTo roles functionally determine the departsThroughGate role, which leads to the gate. In this

example, two flights having the same arrival and departure airports should also agree on the departure gate.

Another interesting use of functional dependencies is related to the notion of *keys*. Consider as an example a Passport relation in the Flight database. Let us assume that an expert in the domain states that the country and pass number are the keys of the passport relation. In an ontology, we can think of Country, Passport and Person as concepts with the roles displayed in Figure 2 linking them:

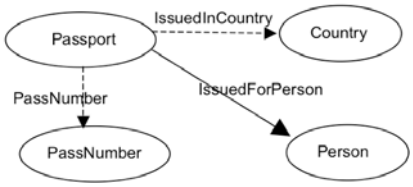

Fig. 2. Passport key in an ontology

Similarly to what we described in databases we could define that the roles issuedInCountry and passNumber compose the keys for a Passport. The roles represented by dotted lines are the ones marked as part of the key. In this case the key would ensure that *"two passports issued for the same country and having the same pass number are the same"*. If they are the same, it is obvious that all the other roles must also agree on their values.

More complex and interesting FDs can be defined over paths of roles. Consider the following example of flight tickets depicted in Figure 3. The price of the flight ticket depends on the arrival and departure airports:

Fig. 3. FD with paths for a flight ticket

In Figure 3 the FD is defined not only based in the roles having Ticket as domain, but also on paths of roles starting from Ticket. Moreover, we can be interested in explicitly stating how exactly the price is determined based on the airports. For instance we could define a function that calculates the price based on the distance between the two airports:

f_{price}(*departureAirport*,*arrivalAirport*) = distance(departureAirport,arrivalAirport)

In that case we explicitly specify the function and that is why we will refer to this case as Explicit Dependency throughout this paper (Figure 4).

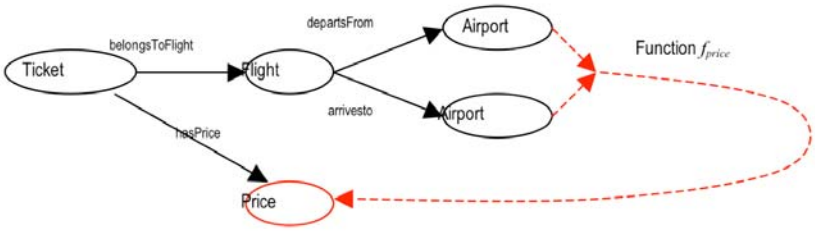

Fig. 4. FD for the flight ticket with explicit function

Up to now we have seen several examples of FD enforcement rules that would add expressivity to ontologies. We can classify them as "classical" FDs like in the ticket price example in Figure 3; key FDs like in the passport example in Figure 2; and FDs with explicit function like in Figure 4.

So we see the need of defining all these flavours of FDs in DL. In the Web Ontology Language OWL-DL[14], only basic FDs are expressible thanks to the `FunctionalProperty` and the `InverseFunctionalProperty`.

In the next section we describe a formal framework that accommodates: classic FDs, keys and explicit dependencies.

4. Formalization Framework

4.1 Abstract Syntax

A FD definition *fd*, used for FD reasoning at the instance level, is composed of the following elements: the antecedent *A*, consequent *C*, a root concept *R* and eventually a skolem function *f*:

$$fd = (A, C, R, f) \quad (1)$$

This definition can also be expressed as an implication, in the same vein as traditional FDs:

$$(fd\ R : A \xrightarrow{f} C) \quad (2)$$

The antecedent *A* is a set of paths. A path u_i is in turn composed of a list of roles, each one being r_i. The consequent is defined by a single path *u*, which is composed of *l* roles u_i. The root concept *R* is the starting point of all paths in the antecedent and consequent, so that a FD expresses relationships among roles of a single instance of the C concept. Notice that all paths considered are single valued and simple

concatenations of roles, such that more complex composition constructs are not allowed.

$$\begin{aligned} A &= \{u_1, u_2, ..., u_n\} \\ u_i &= \{r_{i,1}, r_{i,2}, ..., r_{i,m_i}\} \\ C &= \{u\} \\ u &= \{s_1, s_2, ..., s_l\} \end{aligned} \quad (3)$$

In case of having the deterministic function *f* defined, it takes as parameters individuals of the ranges of the last roles of the antecedent paths. And the result of *f* must be an individual in the range of the last role of the path in the consequent.

4.2 FD Semantics

Concerning the semantics of the *fd* definition, we first define path evaluation under an interpretation X. Given an interpretation X, we say it is composed by a domain Δ^X and an interpretation function. As we have seen the interpretation function maps a role $r_{i,j}$ to a subset $r_{i,j}^X \subseteq \Delta^X \times \Delta^X$. For paths we apply the same principle using composition of these interpretation functions. Given a path u_i, a concept R and an individual x, with $x \in R^X$, then $u_i^X(x)$ is defined as:

$$r_{i,m}^X(\ldots(r_{i,2}^X(r_{i,1}^X(x)))\ldots)$$

Now an interpretation X satisfies a FD $fd = (A, C, R, f)$, with A and C defined as in (1), if for all $a, b \in R^X$ it is verified that:

if $u_1^X(a) = u_1^X(b)$ and
…
$u_i^X(a) = u_i^X(b)$ and
…
$u_n^X(a) = u_n^X(b)$,

then $u^X(a) = u^X(b)$

4.3 Classic FDs

In the simple example of the flight gate that depends on the arrival and departure airports (see Figure 1), the *fd* definition would be composed of the following antecedent, consequent and root concept:

$A = \{u_1, u_2\}$ $C = \{u\}$ $R = \text{Flight}$
$u_1 = \{\text{departsFrom}\}$
$u_2 = \{\text{arrivesTo}\}$
$u = \{\text{departsThroughGate}\}$

We can express FDs as Horn clause rules so that later an engine can enforce the FDs for the instances of an ontology. In the case of classic FD the abstract *fd* definition in (1) can be translated to the following Horn rule:

$$
\begin{aligned}
&r_{1,1}(a, p_{1,1}) \wedge r_{1,2}(p_{1,1}, p_{1,2}) \wedge \ldots \wedge r_{1,m_1}(p_{1,m_1-1}, g_1) \wedge \\
&\ldots \wedge \\
&r_{n,1}(a, p_{n,1}) \wedge r_{n,2}(p_{n,1}, p_{n,2}) \wedge \ldots \wedge r_{n,m_n}(p_{n,m_n-1}, g_n) \wedge \\
&\ldots \wedge \\
&r_{1,1}(b, q_{1,1}) \wedge r_{1,2}(q_{1,1}, q_{1,2}) \wedge \ldots \wedge r_{1,m_1}(q_{1,m_1-1}, g_1) \wedge \\
&\ldots \wedge \\
&r_{n,1}(b, q_{n,1}) \wedge r_{n,2}(q_{n,1}, q_{n,2}) \wedge \ldots \wedge r_{n,m_n}(q_{n,m_n-1}, g_n) \wedge \\
&s_1(a, p_1) \wedge s_2(p_1, p_2) \wedge \ldots \wedge s_l(p_{l-1}, p_l) \wedge \\
&s_1(b, q_1) \wedge s_2(q_1, q_2) \wedge \ldots \wedge s_l(q_{l-1}, q_l)
\end{aligned}
\rightarrow sameAs(p_l, q_l)
$$

The a, b, $p_{i,j}$, q_{ij} *and* g_i elements are free variables. The variables a and b are the common root nodes linking all the paths in the antecedent and consequent of the FD. The $r_{i,j}$ are roles of an antecedent path and the s_i are roles of the consequent, just as shown in (2) (3). This mappings suffers slight variations when applied to the case of key and explicit functions.

4.4 Keys

If the FD represents a key, FD *fdk*, then the consequent is the instance of the root concept itself (*Id*) and there is no need to specify *C*. It is not necessary to specify *f* either:

$$fdk = (A, R)$$

$$(fdk\ R : A \rightarrow Id) \quad (4)$$

Given the interpretation X, it satisfies the key *fdk* if for all $a, b \in R^X$:

if $u_1^X(a) = u_1^X(b)$ and

$\ldots$

$u_i^X(a) = u_i^X(b)$ and

$\ldots$

$u_n^X(a) = u_n^X(b)$,

then $a = b$

Notice that the only difference at the interpretation level is that instead of ensuring the equality between $u^X(a) = u^X(b)$, we need to ensure the equality of the instances a and b themselves.

In the simple example of the passport with a key FD, the *fdk* definition would be composed of the following antecedent and root concept:

$A = \{u_1, u_2\}$ $C = \{u\}$ $R =$ Passport

$u_1 =$ {issuedInCountry}

$u_2 =$ {passNumber}

The *fdk* needs to ensure that the instances are themselves equal if the antecedent holds.

In the case of key FD the abstract *fdk* definition in (4) can be translated to the following Horn rule:

$$
\begin{aligned}
&\mathsf{r}_{1,1}(a, p_{1,1}) \wedge \mathsf{r}_{1,2}(p_{1,1}, p_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,m_1}(p_{1,m_1-1}, g_1) \wedge \\
&\ldots \qquad\qquad \rightarrow \mathit{sameAs}(a,b) \\
&\mathsf{r}_{n,1}(a, p_{n,1}) \wedge \mathsf{r}_{n,2}(p_{n,1}, p_{n,2}) \wedge \ldots \wedge \mathsf{r}_{n,m_n}(p_{n,m_n-1}, g_n) \wedge \\
&\ldots \wedge \\
&\mathsf{r}_{1,1}(b, q_{1,1}) \wedge \mathsf{r}_{1,2}(q_{1,1}, q_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,m_1}(q_{1,m_1-1}, g_1) \wedge \\
&\ldots \wedge \\
&\mathsf{r}_{n,1}(b, q_{n,1}) \wedge \mathsf{r}_{n,2}(q_{n,1}, q_{n,2}) \wedge \ldots \wedge \mathsf{r}_{n,m_n}(q_{n,m_n-1}, g_n)
\end{aligned}
$$

The a, b, $p_{i,j}$,q_{ij} and g_i elements are variables in the rule language.

4.5 Explicit Function

In defining explicit function FDs *fde*, the deterministic function f has to be specified along with the antecedent and consequent:

$$fde = (A, C, R, f)$$

$$(fde\ R : A \xrightarrow{f} C) \quad (5)$$

Given the interpretation X, it satisfies the explicit FD *fde* if for all $a \in \mathrm{R}^{\mathrm{X}}$, and $t_1, \ldots, t_n \in \Delta^{\mathrm{X}}$

if $t_1 = u_1^{\mathrm{X}}(a)$ and
…
$t_i = u_i^{\mathrm{X}}(a)$ and
…
$t_n = u_n^{\mathrm{X}}(a)$,

then $u^{\mathrm{X}}(a) = f(t_1, \ldots, t_i, \ldots, t_n)$

For example in the more complex case of the ticket price we would have:

$A = \{u_1, u_2\}$ $C = \{u\}$ $R = \mathsf{Ticket}$ $f = f_{ticket}$

$u_1 = \{\mathsf{belongsToFlight}, \mathsf{departsFrom}\}$

$u_2 = \{\mathsf{belongstoFlight}, \mathsf{arrivesTo}\}$

$u = \{\mathsf{hasPrice}\}$

Notice that in this example we have two paths u_1 and u_2 each one having two components. The function f_{ticket} takes airports as parameters and returns a price instance.

The abstract *fde* syntax in (5) can be translated to the following Horn rule:

$$
\begin{aligned}
&\mathsf{r}_{1,1}(a, p_{1,1}) \wedge \mathsf{r}_{1,2}(p_{1,1}, p_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,m_1}(p_{1,m_1-1}, g_1) \wedge \\
&\ldots \qquad\qquad \rightarrow \mathsf{s}_l(p_{l-1}, f(g_1, \ldots, g_i, \ldots, g_n)) \\
&\mathsf{r}_{i,1}(a, p_{i,1}) \wedge \mathsf{r}_{i,2}(p_{i,1}, p_{i,2}) \wedge \ldots \wedge \mathsf{r}_{i,m_i}(p_{i,m_i-1}, g_i) \wedge \\
&\ldots \wedge \\
&\mathsf{r}_{n,1}(a, p_{n,1}) \wedge \mathsf{r}_{n,2}(p_{n,1}, p_{n,2}) \wedge \ldots \wedge \mathsf{r}_{n,m_n}(p_{n,m_n-1}, g_n) \wedge \\
&\mathsf{s}_1(a, p_1) \wedge \mathsf{s}_2(p_1, p_2) \wedge \ldots \wedge \mathsf{s}_{l-1}(p_{l-2}, p_{l-1})
\end{aligned}
$$

The a, $p_{i,j}$, and g_i elements are free variables.

Having presented the syntax and semantics for the three FDs modes discussed in this work, we turn now to discussing enforcement policies with respect to a knowledge base, which we name FD interpretations.

4.6 FD Interpretations

An interesting aspect about FDs in ontologies is that depending on the kind of enforcement, they can be applied quite differently. We have identified three FD interpretations: constraints, new assertions and views.

In the first enforcement mode (i.e. constraints) FD expresses invalid states of a ABOX. Instances conforming to an FD constraint are identified and exposed to user analysis. The second interpretation creates new ABOX assertions with instances matching the FD definitions. Finally, view interpretation corresponds to retrieving instances matching FD specifications.

To better understand this difference of usage of FD assertions, consider the following example, again in the context of the 'flight ontology': "*The tax on a ticket price functionally depends on the passenger age-group, the departure airport and the arrival airport*".

We identify the paths for the antecedent and consequent; and the function f_{tax} that computes the tax based on the departure, arrival and age group: $tax = f_{tax}(departureAirport, arrivalAirport, ageGroup)$.

The FD is defined as:

$$fd_{tax} : (A, C, \text{Ticket}, f_{tax})$$
$$A = \left\{ \begin{array}{l} \{\text{belongsToFlight}, \text{departsFrom}\}, \\ \{\text{belongsToFlight}, \text{arrivesTo}\}, \\ \{\text{hasPassenger}, \text{belongsToGroup}\} \end{array} \right\} \quad (6)$$
$$C = \{\{\text{hasPrice}, \text{hasTax}\}\}$$

Consider, in addition, the following ABOX:

belongsToFlight(T1,F1)
departsFrom(F1,GENEVA)
arrivesTo(F1,HEATHROW)
hasPassenger(T1,CARL)
belongsToGroup(CARL,JUNIOR).

The FD assertion interpretation would produce the following ABOX statement hasTax(P1, f_{tax}(GENEVA,HEATHROW,JUNIOR)) for a price 'P1' of ticket 'T1'. Symmetrically, in case of adapting the FD constraint enforcement interpretation, the role hasTax would appear in the consequent of a FD specification in its negative form to check for hurting instances, such as: *not* hasTax(P1, f_{tax}(GENEVA,HEATHROW,JUNIOR)). Finally, view interpretation is syntactically equivalent to FD assertion but with interpretation leading to instances been returned to the user.

4.7 Extended Knowledge Base

In order to accommodate the aforementioned interpretations we extend the conceptual model proposed in [5] according to the following extended DL-FD knowledge base, represented as a sextuple:

$$\mathcal{K}=(\mathcal{T},\mathcal{A}, \mathcal{FD}, \mathcal{C}, \mathcal{C}_{\mathcal{A}}, \mathcal{V})$$

Such that:

$\mathcal{T}$ is a finite set of standard TBox axioms,

$\mathcal{A}$ is a finite set of standard ABox assertions,

$\mathcal{FD}$ is a finite set of functional dependency definition instances, where each FD definition can be classified as:

$\mathcal{FDa}$ is a finite set of assertion FDs fda_i

$\mathcal{FDc}$ is a finite set of constraint FDs fdc_i

$\mathcal{FDv}$ is a finite set of view FDs fdv_i

C is a finite set of constraint witness classes w_{fdci}, with $fdc_i \in \mathcal{FDc}$

$C_{\mathcal{A}}$ is a finite set of assertion hurting some $\mathcal{FDc}$ constraint and expressed as *witness* facts, i.e. instances of w_{fdci} .

$\mathcal{V}$ is a finite set of view definitions

$\mathcal{V}=\{v_i \equiv fdv_i, \ldots, v_n \equiv fdv_n\}$, where $fdv_i \in \mathcal{FDv}$

The set C of witness classes models instances (i.e. $C_{\mathcal{A}}$) hurting FD constraints. They allow users to analyze the hurting instances without directly affecting the ABOX.

The view interpretation specifies queries whose answers are computed by the explicit dependency function over determining property values. The view characterization defers from simple assertions in that the FD rule definition specifies necessary and sufficient conditions for ABox assertions to match with predicates in FD.$\mathcal{V}$ comprehends view labels mapped to corresponding $\mathcal{FDv}$ instances.

Having defined FDs formally and integrated them within an extended knowledge base, we discuss in the next section how functional dependency is specified in OWL.

5. FD Definitions in OWL-DL

In this section, the formalism introduced in section 4 is realized into an approach for integrating FD into OWL-DL.

5.1 OWL FD Package

In order to model the abstract FD definition presented in (1) and (2), an OWL Class called `FD` has been specified. This class, its subclasses and properties, have been defined in an OWL FD package with a separate namespace `owlfd`. In this way, we can reuse these FD definitions in any owl ontology, by importing the `owlfd` namespace:
<owl:imports rdf:resource="http://lbd.epfl.ch/fdowl.owl"/>

5.2 OWL FD Class

The owlfd:FD Class, just like in the definition introduced in (1), has the following properties: antecedent, consequent, rootClass and hasFunction. The antecedent property links FD instances to one or more Path instances. Similarly the consequent property links a FD instance to at most one Path. The rootClass property has a rdf:Class as range and it links a FD to a class name in the OWL ontology. The rootClass reflects the root concept of the abstract FD. Finally, the hasFunction property indicates the resource id of the function corresponding to *f* as in the abstract definition.

```
FD ⊆
owl:Thing
∀antecedent only Path
≥antecedent min 1
∀consequent only Path
≤consequent max 1
=rootClass exactly 1
≤hasFunction max 1
```

For the case of keys, a sub-property of rootClass called keyRootClass has been defined. Any FD definition featuring this subproperty instead of rootClass should be interpreted as a FD key definition.

The Path class, referenced by the antecedent and consequent is a Class that contains a list of property references called owlfd:PartList. The PartList class is a extension of the generic rdf:List, specializing the rdf:first and rdf:last properties. In order to make the PartList an ordered list of references to properties, the "first" property of this list can only accept rdf:Property instances. We give the PartList definition as:

```
PartList ⊆
        rdf:List
        ∀rdf:first only rdf:Property
          =rdf:first exactly 1
        ∀rdf:rest  only rdf:List
          =rdf:rest  exactly 1
```

Now a Path is linked to a PartList through the parts property. A path must have one PartList.

We give now the definition of a Path:

```
Path⊆
     owl:Thing
      ∃parts some PartList
     =parts exactly 1
```

5.3 Subclasses of FD

In addition we have defined three subclasses of FD: FDa, FDc and FDv. These subclasses correspond to the abovementioned interpretation types: assertions, constraints and views respectively:

```
FDa ⊆FD
FDc ⊆FD
```

$$\mathtt{FDv} \subseteq \mathtt{FD}$$

As we have seen in the previous section, these interpretation differences don't have much impact on the abstract definition. In fact it is sufficient to use one of the three aforementioned subclasses (`FDa`, `FDc` or `FDv`) to get the expected results in terms of interpretations.

6. Implementation

Having described our approach for adding functional dependencies to OWL, we proceed now to describe a prototype implementation demonstrating the applicability of our ideas.

6.1 Implementation design

The starting point for implementation of functional dependencies for ontologies is definitely the FD constructs definition. We have described how FDs can be described in abstract terms and how this abstraction can be expressed using our OWL FD classes and properties. It is important to notice that the FD definitions are independent from any actual implementation of the enforcement of the dependencies. The mechanisms to guarantee that the definitions hold could follow various different approaches. On this work we have focused on mapping the FD definitions to Horn clause rules. In the specific case of OWL, the SWRL language constitutes a concrete example of an effort unifying OWL DL and Horn clauses. We have already shown how to map the OWL FD definitions to Rules. This mapping mechanism has been implemented for the three discussed interpretations. FD definitions and derived rules are based on predicates whose terminology is part of a known knowledge base.

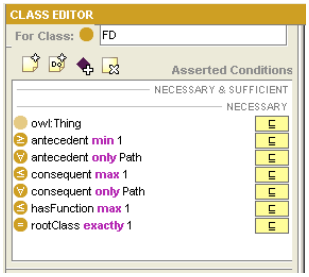

Fig. 5. FD Class in Protégé [23].

Instances of `FD` are functional dependency definitions for the ontology. Figure 6 below illustrates a Protegé OWL FD instance specification.

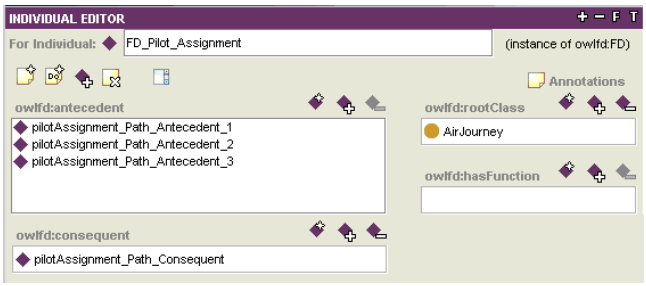

Fig. 6. FD antecedent and consequent

Each `Path`, in the FD_Pilot_assigment with their `PathLists`, is also easily editable with Protégé:

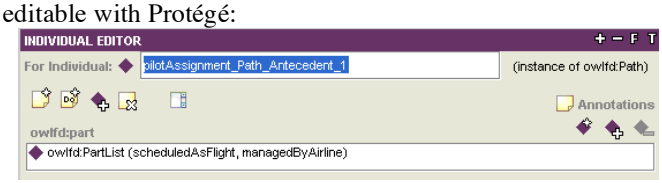

Fig. 7. Path with PartList

In this example, the Path is given by the PartList composed of properties scheduledAsFlight and managedByAirline.

6.2 Mapping from OWL FD to SWRL

We have developed a Java application that takes OWL FD definitions of an ontology and generates the corresponding set of SWRL rules. This procedure follows the mapping described in section 4. In the next subsections, we will reconsider the *tax* example of section 4.6, with the three variants of interpretation. Figure 8 shows a generated SWRL rule in the SWRL tab of Protégé.

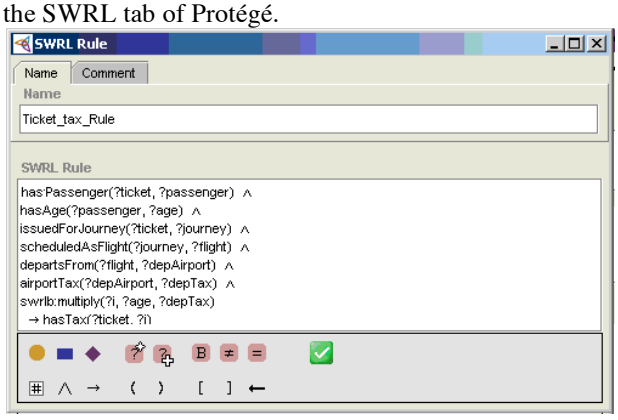

Fig. 8. SWRL rule for tax FD

For the sake of simplicity, in this example the f_{tax} function has been replaced by a simple multiplication function called multiply, which is available out of the box as a SWRL Built-In function. It is supported in the basic package of the SWRL rule engine we used. Nevertheless we could specify a more complex function, with its behavior implemented in a Java class.

In the following sections we present the variations according to the intended interpretation.

6.2.1 Assertion SWRL rules

To differentiate this kind of FD definitions, we use the `FDa` subclass of our `FD` Class. In this first case the head of the rule, or the "result" of the rule evaluation, is a predicate that is added to the ABOX of the knowledge base. This predicate is a property assertion of the kind propertyName*(?variable1,?variable2)*. In the example the propertyName is hasTax, the variable *?ticket*, represents a ticket individual matching the conditions in the rule's body, and the *?i* variable is the result of the evaluation of the swrlb:multiply function over the variables *?age* and *?depTax*. These last two are the age of the passenger of the ticket and the tax of the departure airport. To add the results of the rule evaluation to the ABOX, the user has to export the resulting predicate back to OWL through the Protégé interface.

6.2.2 Constraint SWRL Rules

These FDs are individuals of the subclass `FDc`. Contrary to `FDa` rules, these do not add any new assertions to the ABOX as a result of FD evaluation. Instead, their enforcement checks whether existing ABOX assertions are consistent with the `FDc` definitions. In case of hurting instances are detected, they are classified to the corresponding witness class, which holds the information about the individual who is violating the `FDc` constraint.

A witness property in its most basic form indicates which individual violates the constraint and the expected instance value. For example in the tax example, if for some reason someone has asserted that hasTax(TICKET1,300), this contradicts the expected predicate hasTax(TICKET1,200). The following witness is produced: $\text{witness}_{\text{tax}}$(TICKET1,200). We can see the complete SWRL rule in the Protégé interface, Figure 9.

Notice that the witness can grow in complexity and the information it could eventually hold depends on how the witness property is modeled. This is similar to custom exceptions in a programming language. The witness properties are defined in their own constraint terminology set C, as described in section 4.7. The witness assertions are in turn stored in the $C_{\mathcal{A}}$ set.

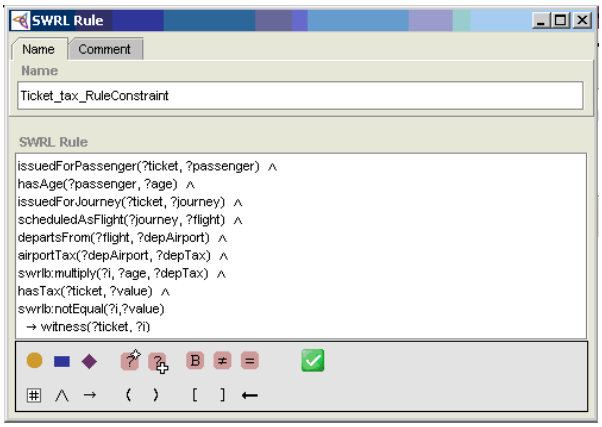

Fig. 9. Constraint SWRL rule

6.2.3 Views with SWRL Rules

As we have already mentioned, the case of views is quite similar to that of new-assertions. The chief difference is that the predicates of the head of the rules, the results of the rule evaluation, are not added to the ABox. They are computed at run-time during query processing. For example in the model of *tax*, equation (6), the ticket tax is computed and retrieved in a query, but never stored anywhere. For views the results are displayed in the context of query execution.

7. Conclusions

The extension of DL knowledge base with functional dependencies has been acknowledged as relevant in producing more expressive ontologies. In this work we investigate the extension of knowledge bases with three kinds of functional dependencies: classic, keys and featuring explicit functions. In fact, to the best of our knowledge, this is the first work in ontologies that explores functional dependencies with an explicit function relating dependent to determining properties. We propose a formal framework to extend ontologies with these three functional dependencies and study the different behaviors that can be considered when running FD as Horn clause rules. We identified three main types of interpretations for FDs: constraints, new-assertions and views; and show how to integrate them within a common structure. The conceptual representation is implemented in OWL by a new OWL FD concept that can be added to any OWL ontology. This concept holds all the attributes of an FD as properties and its instances are called functional dependency definitions. Moreover, a mapping function translates FD assertions into SWRL rules, allowing inferences to

produce the desired FD behavior. The framework has been implemented in an initial prototype under Protégé and using Jess as the rule execution engine.

Our approach to extend the knowledge base with a new FD class has both advantages and disadvantages. An advantage is that it can be easily adopted without requiring any extension to the language. Furthermore, as the FD evaluation is done through SWRL it does not affect subsumption reasoning in the TBox. It turns out that this same aspect can be seen as a disadvantage as subsumption cannot be expressed over constrained concepts with FD.

One of the main problems with functional dependencies and especially keys, is to evaluate equality. A pragmatic option is to define equality based on datatype properties of individuals, but this is a whole subject on its own and may deserve a deeper analysis.

Another interesting issue that we leave for future investigation is the case of key FD with multi-valued non-key attributes. In this scenario deciding on equality of sets seems not evident.

Similarly, if properties in the head of a FD are allowed to be multi-valued then existential quantification over the set is required. In all the models presented in this work, paths and FDs are modeled over single valued properties.

REFERENCES

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

[2] R. Fagin. Functional dependencies in a relational data base and propositional logic. *IBM Journal of Research and Development 21(6)*, pages 543-544. 1977.

[3] Jun Hong, Weiru Liu, David A. Bell, Qingyuan Bai. Answering Queries Using Views in the Presence of Functional Dependencies. *BNCOD 2005*, pages 70-81. 2005.

[4] Serge Abiteboul and Oliver Duschka, Complexity of answering queries using materialized views. In *Proc. Of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Seattle, WA, 1998

[5] B. Motik, I. Horrocks, U. Sattler. Bridging the Gap Between OWL and Relational Databases. In *Proceedings of the 16th international conference on World Wide Web*, pages 807-816, 2007.

[6] D. Toman, G. E. Weddell, On Keys and Functional Dependencies as First-Class Citizens in Description Logics. On *IJCAR* 2006: 647-661, 2006

[7] Beeri C., and Vardi, M. Y. Formal system for tuple and equality generating dependencies. SIAM J Comput 13 (1984), 76--98.

[8] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/ W3C Member Submission 21 May 2004.

[9] A. Borgida and G. E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In *Proceedings of the Fifth International*

Conference on Deductive and Object Oriented Databases, pages 85--102, 1997.

[10] A. Borgida, R. Brachman, D. McGuinness, L. Alperin Resnick. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59-67. June 1989.

[11] V. L. Khizder, D. Toman, and Grant E. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *International Conference on Database Theory ICDT'01*, pages 54-67, 2001.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini. Keys for free in Description Logics. In *Proc. of the 2000 International Workshop on Description Logics (DL'2000)*, 2000.

[13] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider, Eds. The *Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[14] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein. OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/, W3C Recommendation 10-02-2004.

[15] D. Berardi, D. Calvanese, G. De Giacomo. Reasoning on UML class diagrams. In *Artificial Intelligence Volume 168, Issues 1-2*. October 2005, pages 70-118.

[16] H. Boley, S. Tabet, G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proceedings of SWWS'01*, Stanford. 2001.

[17] D. Calvanese, G. De Giacomo, M. Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proc. of IJCAI 2001*, pages 155--160, 2001.

[18] R. Fagin. Horn Clauses and Database Dependencies. In *Journal of the Association for computing Machinery*, Vol 29, no 4, pages 952-985, 1982.

[19] R. Fagin. Normal Forms and Relational Database Operators. *ACM SIGMOD International Conference on Management of Data, May 31-June 1*, 1979, Boston, Mass. Also IBM Research Report RJ2471, Feb. 1979.

[20] C. Lutz, C. Areces, I. Horrocks, U. Sattler. Keys, Nominals and Concrete Domains. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 349-354. Morgan Kaufmann. 2003.

[21] C. Lutz, M. Milicic, Description Logics with Concrete Domains and Functional Dependencies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, 2004.

[22] B. Ludascher, A. Gupta, M. E. Martone, Model-Based Mediation with Domain Maps, *17th International Conference on Data Engineering*, Heidelberg, Germany, 2001.

[23] Protégé Community, SWRLQueryBuiltIns. Protégé Wiki http://protege.cim3.net/cgi-bin/wiki.pl?SWRLQueryBuiltIns, July 2007.

[24] D. Toman, G. E. Weddell. On Path-functional Dependencies as First-Class citizens in *the 2005 International Workshop on Description* Description Logics. On *Proceedings of Logics (DL2005)*, Edinburgh, Scotland, UK, July 26-28, 2005.