Functional Dependencies in OWL ABoxes

Jean-Paul Calbimonte[1], Fabio Porto[2], C. Maria Keet[3]

[1]École Polytechnique Fédérale de Lausanne (EPFL)
Database Laboratory - Switzerland

jean-paul.calbimonte@epfl.ch

[2]National Laboratory of Scientific Computation (LNCC)
Computer Science Coordination – Petropolis, Brazil

fporto@lncc.br

[3]Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

keet@inf.unibz.it

Abstract. *Functional Dependency has been extensively studied in database theory. Most recently, there have been some works investigating the implications of extending Description Logics with functional dependencies. As it turns out, more complex functional dependencies at the type-level can lead to undecidability, which thus restricts its usage in the TBox. This paper therefore focuses on enhancing its applicability to instances in the ABox. We specify 'FD' as a new constructor, realized as 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 in the knowledge base. Our approach thereby gives ontology users data guarantees and features usually found only in databases.*

1. Introduction

Data dependencies have been introduced as a general formalism for a large class of database constraints that augments the expressivity of database schemas [1]. Functional dependencies (FD) are a particularly interesting type of data dependency [2] that elegantly capture relationships between attributes of a relation leading to the identification of primary keys and is used for the normalization procedure of a conceptual or logical data model in order to avoid redundancy in representation of the data. Other important applications of FD in database include query rewriting [3] and query evaluation [4].

The semantics expressed through functional dependencies are equally relevant when specifying a conceptual model by means of an application ontology for ontology-driven information systems. 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 that contains assertions about individuals of the vocabulary in the TBox.

Indeed, in recent years, a bulk of prior research has investigated the implications of adding functional dependencies to ontology languages (e.g. [6,9,11,12,19,20]). These initiatives took one of two paths: extending a DL language with a new FD concept constructor or adding FD (and key) as number restrictions 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 (TBox) and remaining in the decidable fragment of first order logic requires limiting the expressiveness of a DL language in various ways. For many data-centric ontology-driven applications, however, correctness of entered data in the ABox may be more relevant than the expressiveness at the type-level. This work takes the latter assumption and proposes an extension of DL ontologies with database-like expressive FD assertions. Indeed, FDs are specified as instances of a newly defined TBox concept named "FD". The here proposed solution for FD assertions in an OWL-ontology setting—to meet user' requirements, adhere to W3C standards, and circumvent certain theoretical limitations—allows formulating complex FD rules, including multiple paths in both the antecedent and consequent of the rule. Three types of FDs will be 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 realize ABox FDs by mapping FD instances to Horn clauses [17,18] using the SWRL rule language [8]. 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: FDs interpreted as constrains, as assertions and as views. Firstly, the constraint behavior indicates instances that do not comply with the FD rules. The 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 remainder of this paper is structured as follows. Section 2 contains the preliminaries with related works and problem motivation using examples. Section 3 presents a formal framework for the FD construct and discusses enforcement interpretation. Section 4 introduces the FD construct in OWL and section 5 presents a first prototype implementation. Finally, we conclude in section 6.

2. Motivating examples and related work

We first sketch FD functionality desired by modelers and discuss relevant theoretical contributions and limitations afterwards.

2.1 Motivating examples

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 $X \square Y$, 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 (values of the) attributes in X. When transposing similar rules to the ontology world we discover that FDs could indeed be very useful to enrich the representation of subject domain information. Take, for instance, an ontology about flights. We can partially model the Flight, Airport and Gate concepts and their linking roles, as shown in Figure 1.

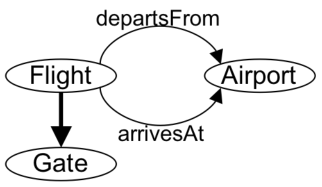


Fig. 1. The FD *departsThrough* for *Flight* and *Gate*, indicated with a thick arrow.

In this representation, the *departsFrom* and *arrivesAt* roles functionally determine the *departsThrough* 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* concept in the Flight ontology. Let us assume that an expert in the domain states that the *Country* and *PassNumber* compose the keys of the *Passport* concept, i.e., *Passport* is a weak entity type. Similarly to what one would express in a database schema we could specify that the roles *issuedInCountry* and *PassNumber* compose the key for a *Passport*. In an ontology, we can think of *Country*, *Passport* and *Person* as concepts with the roles displayed in Figure 2 establishing a relationship between them. 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.

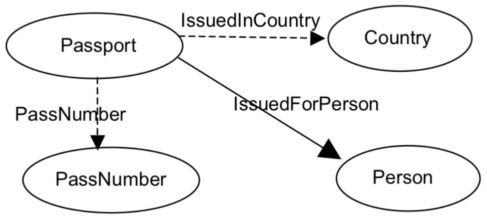


Fig. 2. The key for the weak entity type *Passport* in an ontology; participating roles are drawn with dashed lines.

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

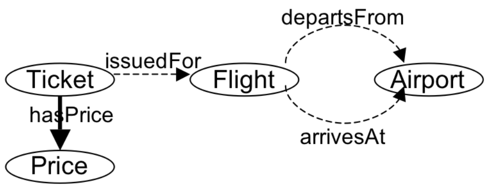


Fig. 3. FD with paths for a flight ticket.

In Figure 3, the FD is defined not only based on 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).

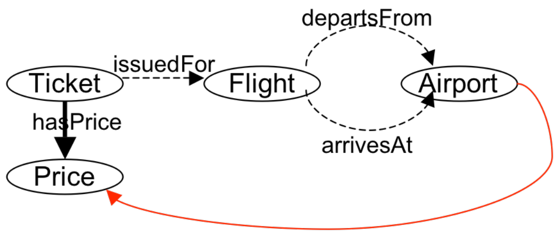


Fig. 4. FD for the flight ticket with explicit function

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. Therefore, we see the need of defining all these flavours of FDs in DL. In the Web Ontology Language OWL-DL [14], as well as its proposed successor OWL 2 DL (based on SROIQ [24]), only basic FDs over a binary relationship are expressible using *FunctionalProperty* and *InverseFunctionalProperty*.

2.2 Related works

Functional dependencies have been extensively studied in databases as a formalism to extend database schema semantics [1,17]. 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 [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 $\mathcal{DLFD}$ is proposed. The authors explored the complexity of logical implication problems in $\mathcal{DLFD}$, by proving equivalence with query answering in Datalog$_{nS}$ with some restrictions, leading to a polynomial time query evaluation.

Calvanese, De Giacomo and Lenzerini, interested in modeling conceptual data models such as ER and UML, as DL knowledge bases, proposed identification and FD assertions for the $\mathcal{DLR}_{ifd}$ language [12] in addition to other common modeling characteristics of conceptual data models, such as n-ary relationships. FDs and uniqueness constraints are mapped to $\mathcal{DLR}_{ifd}$ number restrictions and showed that reasoning with these (non-unary) *fd* assertions is EXPTIME complete. Another interesting feature of $\mathcal{DLR}_{ifd}$ is its ability for representing Object-Oriented class operations (methods) using the *fd* construct [15]. An operation has the form $f(P_1,\ldots,P_h)$:R, where f is the name of the operation, h parameters, each one belonging to classes $P_1,\ldots, P_h$, and the result of f belongs to class R. Formally, such an operation corresponds to a (h+1)-ary predicate; let R be an (h+1)-ary predicate, and $i, \ldots, h, j$ denote components of R, then an interpretation $\mathcal{I}$ satisfies the assertion (fd $R\ i_1, \ldots i_h \rightarrow j$) if for all $t, s \in R^{\mathcal{I}}$, we have that $t[i_1] = s[i_1], \ldots, t[i_h] = s[i_h]$ implies $t[j] = s[j]$. However, $\mathcal{DLR}_{ifd}$ with such FDs has not been implemented in any modeling tool or reasoner.

Lutz et al. [19] 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 also explored the possibility of adding not only keys but also FDs to DLs with concrete domains [20]. 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 problematic (from the perspective of scalable implementations) in the language they defined, $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$.

In [23], 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 ($\sqsubseteq$). However, this extension in the general case is shown to lead to undecidability. Decidability is regained by focusing on a reduced DL where Path FDs occur only at top level or in monotone concept constructors.

Thus, one can observe a clear compromise between expressivity of FDs and the decidability of the resulting DL language. Put differently, the ontology developer's desired FD behaviour as described in section 2.1, (i) *is not met* in present ontology development software and (ii) *might* be implemented only *partially* at the type-level (TBox) (iii) but, despite the theoretical and software limitations, developers still would like to see such functionality *soon*. To address these problems, we necessarily depart from the above-discussed approaches by introducing FD as an *application level construct in the ontology*, i.e. without changing the ontology language, and defer part of the processing to outside of the ontology, where the obtained derived results can be ported back into the ontology. This solution is in part inspired by [5] that elegantly discuss the role of constraints in ontologies as compared to those in databases and the notion of distinguishable witness predicate for holding instances not conforming to specified constraints [21]. We describe a formal framework that accommodates classic FDs, keys, and explicit dependencies in the next section.

3. Formalization Framework

3.1 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 (see formulae 1 and 5):

$$fd = (A, C, R, f) \tag{1}$$

which can be expressed as an implication, in the same vein as traditional FDs:

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

As illustrated in formula 3 below, 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 R concept. Notice that all paths considered are single valued and simple concatenations of roles, such that more complex composition constructs are not allowed.

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

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.

3.2 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:

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

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:

$$\textit{if}\;\; u_1^X(a) = u_1^X(b) \;\textit{and}\;\; u_i^X(a) = u_i^X(b) \;\textit{and}\; \ldots\; u_n^X(a) = u_n^X(b),\; \textit{then}\; u^X(a) = u^X(b)$$

3.3 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:

$$\begin{aligned} & A = \{u_1, u_2\} \;\; C = \{u\} \;\; R = \mathsf{Flight} \\ & u_1 = \{\mathsf{departsFrom}\} \\ & u_2 = \{\mathsf{arrivesAt}\} \\ & u = \{\mathsf{departsThroughGate}\} \end{aligned}$$

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

$$\left.\begin{aligned} & \mathsf{r}_{1,1}(a, p_{1,1}) \wedge \mathsf{r}_{1,2}(p_{1,1}, p_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,\mathrm{m}_1}(p_{1,m_1-1}, g_1) \wedge \\ & \ldots \wedge \\ & \mathsf{r}_{\mathrm{n},1}(a, p_{n,1}) \wedge \mathsf{r}_{\mathrm{n},2}(p_{n,1}, p_{n,2}) \wedge \ldots \wedge \mathsf{r}_{\mathrm{n,m_n}}(p_{n,m_n-1}, g_n) \wedge \\ & \ldots \wedge \\ & \mathsf{r}_{1,1}(b, q_{1,1}) \wedge \mathsf{r}_{1,2}(q_{1,1}, q_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,\mathrm{m}_1}(q_{1,m_1-1}, g_1) \wedge \\ & \ldots \wedge \\ & \mathsf{r}_{\mathrm{n},1}(b, q_{n,1}) \wedge \mathsf{r}_{\mathrm{n},2}(q_{n,1}, q_{n,2}) \wedge \ldots \wedge \mathsf{r}_{\mathrm{n,m_n}}(q_{n,m_n-1}, g_n) \wedge \\ & \mathsf{s}_1(a, p_1) \wedge \mathsf{s}_2(p_1, p_2) \wedge \ldots \wedge \mathsf{s}_\mathrm{l}(p_{l-1}, p_l) \wedge \\ & \mathsf{s}_1(b, q_1) \wedge \mathsf{s}_2(q_1, q_2) \wedge \ldots \wedge \mathsf{s}_\mathrm{l}(q_{l-1}, q_l) \end{aligned}\right. \rightarrow sameAs(p_l, q_l)$$

where 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 $\mathsf{r}_{i,j}$ are roles of an antecedent path and the s_i are roles of the consequent, just as shown in (2) (3). These mappings suffer slight variations when applied to the case of key and explicit functions.

3.4 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:

$$fdk = (A, R)$$

$$(fdk\ R : A \Box\ Id) \qquad (4)$$

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* … $u_i^X(a) = u_i^X(b)$ *and* … $u_n^X(a) = u_n^X(b)$, *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:

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

$$u_1 = \{\text{issuedInCountry}\}$$

$$u_2 = \{\text{passNumber}\}$$

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:

$$\begin{array}{l}
\mathsf{r}_{1,1}(a, p_{1,1}) \wedge \mathsf{r}_{1,2}(p_{1,1}, p_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,\mathsf{m}_1}(p_{1,m_1-1}, g_1) \wedge \\
\ldots \\
\mathsf{r}_{\mathsf{n},1}(a, p_{n,1}) \wedge \mathsf{r}_{\mathsf{n},2}(p_{n,1}, p_{n,2}) \wedge \ldots \wedge \mathsf{r}_{\mathsf{n},\mathsf{m}_\mathsf{n}}(p_{n,m_n-1}, g_n) \wedge \quad \rightarrow \quad sameAs(a,b) \\
\ldots \wedge \\
\mathsf{r}_{1,1}(b, q_{1,1}) \wedge \mathsf{r}_{1,2}(q_{1,1}, q_{1,2}) \wedge \ldots \wedge \mathsf{r}_{1,\mathsf{m}_1}(q_{1,m_1-1}, g_1) \wedge \\
\ldots \wedge \\
\mathsf{r}_{\mathsf{n},1}(b, q_{n,1}) \wedge \mathsf{r}_{\mathsf{n},2}(q_{n,1}, q_{n,2}) \wedge \ldots \wedge \mathsf{r}_{\mathsf{n},\mathsf{m}_\mathsf{n}}(q_{n,m_n-1}, g_n)
\end{array}$$

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

3.5 Explicit Function

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

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

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

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

if $t_1 = u_1^X(a)$ *and* … $t_i = u_i^X(a)$ *and* … $t_n = u_n^X(a)$, *then* $u^X(a) = f(t_1, \ldots, t_i, \ldots, t_n)$

In the more complex case of the ticket price we would have:

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

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

$$u_2 = \{\text{belongstoFlight}, \text{arrivesAt}\}$$

$$u = \{\text{hasPrice}\}$$

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:

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

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

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

3.6 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 the 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:

$$fd_{tax} : (A, C, \text{Ticket}, f_{tax})$$

$$A = \left\{ \begin{array}{c} \{\text{belongsToFlight,departsFrom}\}, \\ \{\text{belongsToFlight,arrivesAt}\}, \\ \{\text{hasPassenger,belongsToGroup}\} \end{array} \right\} \quad (6)$$

$$C = \{\{\text{hasPrice,hasTax}\}\}$$

Consider, in addition, the following ABox:

belongsToFlight(T1,F1)
departsFrom(F1,GENEVA)
arrivesAt(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 being returned to the user.

3.7 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:

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

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:

$\mathcal{FD}_a$ is a finite set of assertion FDs $\mathcal{fda}_i$

$\mathcal{FD}_c$ is a finite set of constraint FDs $\mathcal{fdc}_i$

$\mathcal{FD}_v$ is a finite set of view FDs $\mathcal{fdv}_i$

C is a finite set of constraint witness classes w_{fdci}, with $\mathcal{fdc}_i \in \mathcal{FD}_c$

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

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

$\mathcal{V}=\{v_1 \equiv \mathcal{fdv}_1, \ldots, v_n \equiv \mathcal{fdv}_n\}$, where $\mathcal{fdv}_i \in \mathcal{FD}v$

The set $C_{\mathcal{A}}$ of witness classes models instances 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 $\mathcal{FD}_v$ instances.

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

4. Specifying FD in OWL-DL

In this section, the formalism introduced in section 3 is realized into an approach for integrating FD into OWL-DL.

4.1 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"/>

4.2 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 associating a *FD* instance 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 contains a list of property references called *owlfd:PartList*. The *PartList* class is an 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. The *PartList* definition is specified as:

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

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
```

4.3 Subclasses of FD

In addition three subclasses of *FD* have been defined: FD_a, FD_c and FD_v. These subclasses correspond to the abovementioned interpretation types: assertions, constraints and views respectively:

$FD_a \subseteq FD$

$FD_c \subseteq FD$

$FD_v \subseteq FD$

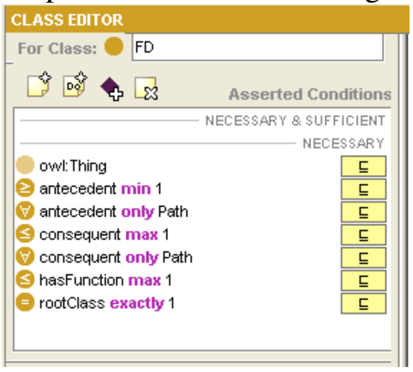
As we have seen in the previous section, these interpretation differences don’t have much impact on the abstract definition. In fact it is sufficient to use one of the three aforementioned subclasses (FD_a, FD_c or FD_v) to get the expected results in terms of interpretations.

5. 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.

5.1 Implementation design

The starting point for implementation of functional dependencies for ontologies is definitely the *FD* constructs definition. We have described how *FD*s can be described in abstract terms and how this abstraction can be expressed using our *OWL FD* classes and properties (see Figure 5). 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. In 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.


CLASS EDITOR

For Class: FD

Asserted Conditions

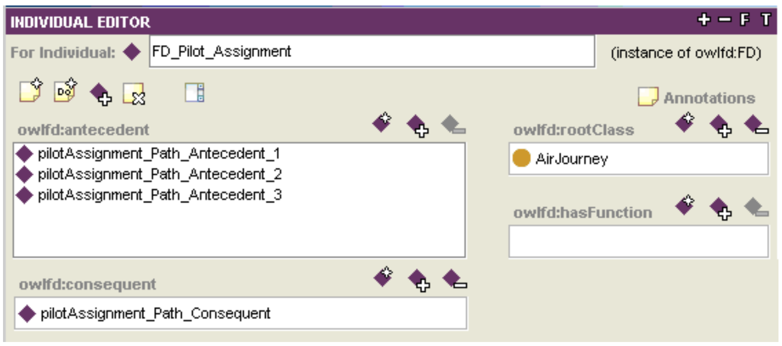
NECESSARY & SUFFICIENT

NECESSARY

owl:Thing ⊑
antecedent min 1 ⊑
antecedent only Path ⊑
consequent max 1 ⊑
consequent only Path ⊑
hasFunction max 1 ⊑
rootClass exactly 1 ⊑

Fig. 5. FD Class in the ontology development tool Protégé.

Instances of FD are functional dependency definitions for the ontology; Figure 6 illustrates a Protégé OWL FD instance specification. Then, each path, such as *FD_PilotAssigment* with their *PathLists* in the Flight ontology, is also easily editable with Protégé. In this example, the *Path* is given by the *PartList* composed of properties *scheduledAsFlight* and *managedByAirline* (see Figure 7).


INDIVIDUAL EDITOR

For Individual: FD_Pilot_Assignment (instance of owlfd:FD)

Annotations

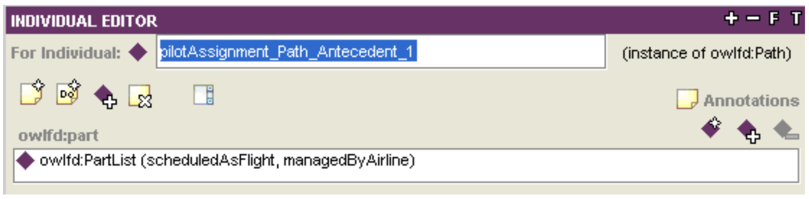
owlfd:antecedent
pilotAssignment_Path_Antecedent_1
pilotAssignment_Path_Antecedent_2
pilotAssignment_Path_Antecedent_3

owlfd:rootClass
AirJourney

owlfd:hasFunction

owlfd:consequent
pilotAssignment_Path_Consequent

Fig. 6. FD antecedent and consequent.



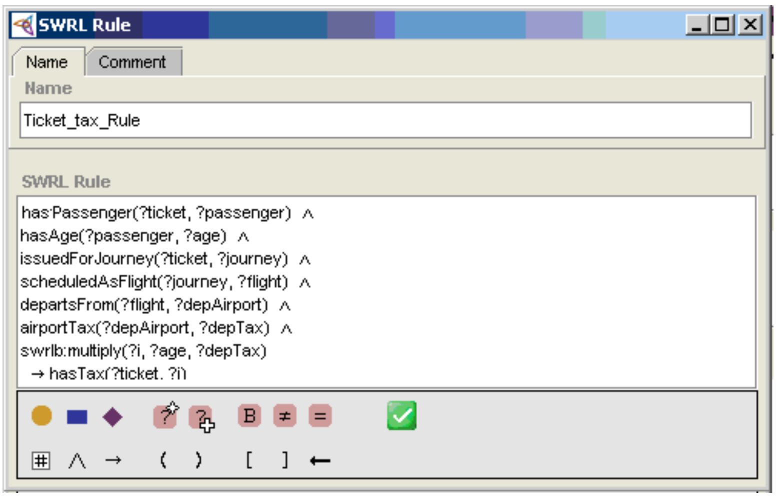
pilotAssignment_Path_Antecedent_1 (instance of owlfd:Path)

owlfd:part
owlfd:PartList (scheduledAsFlight, managedByAirline)

Fig. 7. Path with PartList.

5.2 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 3. In the next subsections, we will reconsider the *tax* example of section 3.6, with the three variants of interpretation. Figure 8 shows a generated SWRL rule in the SWRL tab of Protégé [22].



Ticket_tax_Rule

```
hasPassenger(?ticket, ?passenger) ∧
hasAge(?passenger, ?age) ∧
issuedForJourney(?ticket, ?journey) ∧
scheduledAsFlight(?journey, ?flight) ∧
departsFrom(?flight, ?depAirport) ∧
airportTax(?depAirport, ?depTax) ∧
swrlb:multiply(?i, ?age, ?depTax)
 → hasTax(?ticket, ?i)
```

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 and is supported in the basic package of the SWRL rule engine we used. Alternatively, we could have specified a more complex function and have implemented the intended behaviour using a *Java* class.

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

5.2.1 Assertion SWRL rules

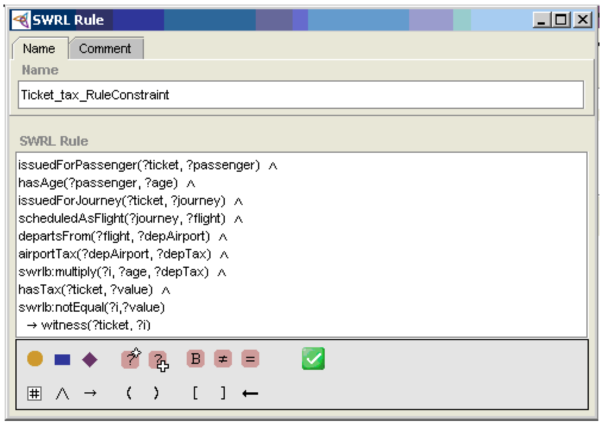
To differentiate this kind of *FD* definitions, we use the FD_a subclass of our FD class. In this first case the head of the rule, or the deduction of the rule evaluation, is a predicate that is added to the ABox of the knowledge base. This predicate is a property assertion of the kind *propertyName(?variable1,?variable2)*. 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 holds 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.

5.2.2 Constraint SWRL Rules

These *FD*s are individuals of the subclass FD_c. Contrary to FD_a 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 FD_c 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 FD_c constraint.

A witness property in its most basic form indicates which individual violates the constraint and the expected instance value. 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: *witness*$_{tax}$*(TICKET1,200)*. We can see the complete SWRL rule in the Protégé interface in 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 3.7. The witness assertions are in turn stored in the $C_{\mathcal{A}}$ set.



Ticket_tax_RuleConstraint

```
issuedForPassenger(?ticket, ?passenger) ∧
hasAge(?passenger, ?age) ∧
issuedForJourney(?ticket, ?journey) ∧
scheduledAsFlight(?journey, ?flight) ∧
departsFrom(?flight, ?depAirport) ∧
airportTax(?depAirport, ?depTax) ∧
swrlb:multiply(?i, ?age, ?depTax) ∧
hasTax(?ticket, ?value) ∧
swrlb:notEqual(?i,?value)
 → witness(?ticket, ?i)
```

Fig. 9. Constraint SWRL rule.

5.2.3 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.

6. 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 ontology language. Furthermore, as the FD evaluation is done through SWRL on instances in the ABox, 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 addition to the paths and FDs that we have modeled over single valued properties in this paper. 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.

7. 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] J. Hong, W. Liu, D.A. Bell, Q. Bai. Answering Queries Using Views in the Presence of Functional Dependencies. In: *Proceeding of BNCOD 2005*, pages 70-81. 2005.

[4] S. Abiteboul and O. 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. In: *Proceedings of IJCAR* 2006: 647-661, 2006

[7] C.Beeri, and M. Y.Vardi. 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, and M. Lenzerini. Identification constraints and functional dependencies in Description Logics. In *Proc. of IJCAI 2001*, 155-160.

[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, and 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, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proceedings of SWWS'01*, Stanford. 2001.

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

[18] 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.

[19] C. Lutz, C. Areces, I. Horrocks, and 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.

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

[21] B. Ludäscher, A. Gupta, and M.E. Martone, Model-Based Mediation with Domain Maps, *17th Int'l Conference on Data Engineering*, Heidelberg, Germany, 2001.

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

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

[24] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In: *Proceedings of KR-2006*, Lake District, UK, 2006.