Translating XML Web Data into Ontologies

Yuan An and John Mylopoulos

University of Toronto, Canada
{yuana,jm}@cs.toronto.edu

Abstract. Translating XML data into ontologies is the problem of finding an instance of an ontology, given an XML document and a specification of the relationship between the XML schema and the ontology. Previous study [8] has investigated the *ad hoc* approach used in XML data integration. In this paper, we consider to translate an XML web document to an instance of an OWL-DL ontology in the Semantic Web. We use the semantic mapping discovered by our prototype tool [1] for the relationship between the XML schema and the ontology. Particularly, we define the *solution* of the translation problem and develop an algorithm for computing a *canonical solution* which enables the ontology to answer queries by using data in the XML document.

1 Introduction

XML has become an accepted standard for publishing data on the Web. To integrate XML data, a former paper [8] has studied the *ad hoc* approach to translating various XML documents into a central ontology instance. In this paper, we study a generic and a formal framework for translating an XML document into an instance of an ontology. The following example illustrates the problem. Suppose we have an XML document $\mathcal{X}$:

```
<db>
   <student sname=''Jerry''>
      <takes>
         <course title=''Database Theory''/>
         <course title=''Combinatorial Optimization''/>
      </takes>
      <advisor pname=''John''/>
   </student>
</db>
```

Suppose we have an ontology shown graphically in Figure 1 using UML notation. Given a natural mapping semantically relating the XML schema to the ontology, we would expect that an instance of the ontology contains the following assertions: *Student*(t_1), *Course*(t_2), *Course*(t_3), *Professor*(t_4), *hasName*(t_1,"Jerry"), *hasTitle*(t_2,"Data base Theory"), *hasTitle*(t_3,"Combinatorial Optimization"), *hasName* (t_4,"John"), *takes*(t_1,t_2), *takes* (t_1,t_3), *hasAdvisor*(t_1,t_4), *Professor*(u_1), *Professor*(u_2), *Course*(u_3), *teaches*(u_1,t_2), *teaches*(u_2, t_3), *teaches*(t_4,u_3), where t_i $i = 1, ..., 4$ and u_j $j = 1, ...3$ are anonymous individuals in the ontology.

Note that the difference between t_is and u_js is that the original XML document provides no information about the individuals u_1, u_2 and u_3. They were deduced by the ontology constraints. However, if we replace u_1 and u_2 by t_4, then the resulting instance will still satisfy all constraints and it says that professor t_4 teaches both courses t_2 and t_3. Alternatively, we could also construct an instance in which the professor t_4 teaches only the course t_2, while the course t_3 is taught by some unknown professor u_2. This tells us that there could be different instances that are consistent with the ontology and satisfy a given mapping from the XML schema to the ontology. So if we are given a source document $\mathcal{X}$ shown above and a query over the ontology, how can we answer it? If our query is, for example, *What is the name of the person who is the advisor of the person whose name is Jerry?* The answer is *John* regardless of a particular instance that was created for the ontology. As another example, consider the query *What is the title of the course taught by Jerry's advisor?* This query cannot be answered with certainty in this scenario.

Ontologies play a central role in the Semantic web. Recently, W3C has recommended the OWL web ontology language for describing ontologies in the Semantic Web. If an XML document needs to be translated into an OWL ontology, the resulting ontology should preserve the information in the XML document and be able to answer queries by using these information. Consequently, a translation involves specifying a mapping, checking the consistency, and preserving information. In this paper, we consider the OWL-DL ontology language because of its close relationship with Description Logics. As a result, the OWL-DL ontology language enable us to develop a translation algorithm. The overall framework is generic in the sense that the theoretical issues apply to many translation problems between databases and ontologies.

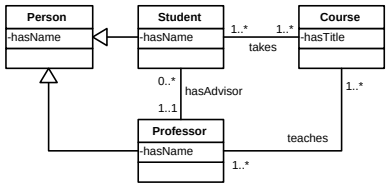

Fig. 1. An Ontology

The rest of the paper is organized as follows. Section 2 presents the formal specifications about OWL-DL ontology and XML. Section 3 defines the problem and Section 4 defines the canonical solution. Section 5 develops the algorithm, and finally, Section 6 gives the conclusions.

2 Preliminaries

We assume readers are familiar with the standard notations and semantics of Description Logics, though we summarize here one flavor relating to the OWL-DL web ontology language. OWL-DL is closely related to the $\mathcal{SHOIN}(\mathbf{D})$ description logic [5], and the meanings of its terminology can be found in [4, 5].

A datatype theory $\mathbf{D}$ is a mapping from a set of datatypes to a set of values. The datatype (or concrete) domain, written $\Delta^{\mathcal{I}}_{\mathbf{D}}$, is the union of the mappings

of the datatypes. Let $\mathbf{R}$ be set of role names consisting of a set of *abstract role names* $\mathbf{R}_A$ and *a set of concrete role names* $\mathbf{R}_D$. The set of $\mathcal{SHOIN}$-roles (or roles for short) consist of a set of *abstract roles* $\mathbf{R}_A \cup \{R^- | R \in \mathbf{R}_A\}$ and a set of *concrete roles* $\mathbf{R}_D$. An *RBox* $\mathcal{R}$ consists of a finite set of transitivity axioms $\mathsf{Trans}(R)$, and role inclusion axioms of the form $R \sqsubseteq S$ and $T \sqsubseteq U$, where R and S are abstract roles, and T and U are concrete roles. $\sqsubseteq^*$ denotes the reflexive-transitive closure of $\sqsubseteq$ on roles, i.e., for two abstract roles R, S, $S \sqsubseteq^* R \in \mathcal{R}$ if S and R are the same, $S \sqsubseteq R \in \mathcal{R}$, $\mathsf{Inv}(S) \sqsubseteq \mathsf{Inv}(R) \in \mathcal{R}$, or there exists some role Q such that $S \sqsubseteq^* Q \in \mathcal{R}$ and $Q \sqsubseteq^* R \in \mathcal{R}$. A role not having transitive sub-roles is called a *simple* role, and $\mathsf{Inv}(R) = R^-$.

The set of $\mathcal{SHOIN}(\mathbf{D})$ concepts is defined by the following syntactic rules, where C_is are concepts, A is an atomic concept, R is an abstract role, S is an abstract *simple role*, T is a concrete role, o_i are individuals, D is a datatype, and n is a non-negative integer:

$$C \rightarrow A | \neg C | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \exists R.C | \forall R.C | \geq nS | \leq nS |$$
$$\{o_1, .., o_n\} | \geq nT | \leq nT | \exists T.D | \forall T.D$$

A *TBox* $\mathcal{T}$ consists of a finite set of concept inclusion axioms $C_1 \sqsubseteq C_2$; an *ABox* $\mathcal{A}$ consists of a finite set of concept and role assertions and individual (in)equalities $C(a)$, $R(a, b)$, $a = b$, $a \neq b$, respectively. A $\mathcal{SHOIN}(\mathbf{D})$ knowledge base (an ontology) $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$, and an ABox $\mathcal{A}$. The semantics of $\mathcal{SHOIN}(\mathbf{D})$ is given by means of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of an non-empty domain $\Delta^{\mathcal{I}}$, disjoint from the datatype domain $\Delta^{\mathcal{I}}_{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$, which interprets atomic and complex concepts, roles, axioms, and assertions in the standard description logic way. An interpretation $\mathcal{I}$ is a *model* of the knowledge base $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ if $\mathcal{I}$ satisfies every concept, axiom, and assertion in $\mathcal{O}$. From the database perspective, the TBox $\mathcal{T}$ and the RBox $\mathcal{R}$ can be viewed as a schema with unary and binary relational tables, and the ABox $\mathcal{A}$ can be viewed as an instance. An ABox $\mathcal{A}$ is *consistent* with respect to $\mathcal{O}$ if there is a model of $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ (we say $\mathcal{O}$ is consistent). A concept or role assertion β is a *logical consequence* of an ABox $\mathcal{A}$ (written $\mathcal{A} \models \beta$), if for every model of $\mathcal{A}$ w.r.t $< \mathcal{T}, \mathcal{R} >$, β is true. We write $\mathcal{O} \models \beta$ for β is a *logical consequence* of the ontology.

An XML document is typically modeled as a node-labeled tree. For our purpose, we assume that each XML document is described by an XML schema consisting of a set of element and attribute type definitions. Specifically, we assume the following countably infinite disjoint sets: **Ele** of element names, **Att** of attribute names, and **Dom** of simple type names including the built-in XML schema datatypes. Attribute names are preceded by a "@" to distinguish them from element names. Given finite sets $E \subset$**Ele** and $A \subset$**Att**, a XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ specifies the type of each element ℓ in E, the attributes that ℓ has, and the datatype of each attribute in A. Specifically, An element type τ is defined by the grammar $\tau ::= \epsilon | \mathsf{Sequence}[\ell_1 : \tau_1, ...\ell_n : \tau_n] | \mathsf{Choice}[\ell_1 : \tau_1, .., \ell_n : \tau_n]$ $(\ell_1, .., \ell_n \in E)$, where ϵ is for the empty type, and $\mathsf{Sequence}$ and Choice are complex types. Each element associates an occurrence constraint with two values:

minOccurs indicating the minimum occurrence and *maxOccurs* indicating the maximum occurrence. The set of attributes of an element $\ell \in E$ is defined by the function $\rho : E \rightarrow 2^A$; and the function $\kappa : A \rightarrow$**Dom** specifies the datatypes of attributes in A. Each datatype name associates with a set of values in a domain *Dom*. In this paper, we do not consider the *simple type elements* (corresponding to DTD's **PCDATA**), assuming instead that they have been represented using attributes. Furthermore, a special element $\underline{r} \in E$ is the root of the XML schema such that $\rho(\underline{r}) = \emptyset$, and we assume that for any two element $\ell_i, \ell_j \in E$, $\rho(\ell_i) \cap \rho(\ell_j) = \emptyset$.

An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ over (E, A) consists of a set of nodes N, a child relation $<$ between nodes, a root node $\underline{r}$, and two functions such as:

- a labeling function λ:$N \rightarrow E \cup A$ such that if $\lambda(v) = \ell \in E$, we say that v is in the element type ℓ; if $\lambda(v) = @a \in A$, we say that v is an attribute $@a$;
- a partial function η:$N \rightarrow Dom$ for every node v with $\lambda(v) = @a \in A$, assigning values in domain *Dom* that supplies values to simple type names in **Dom**.

An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ conforms to a schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$, denoted by $\mathcal{X} \models \mathcal{S}$, if:

1. for every node v in $\mathcal{X}$ with children $v_1, .., v_m$ such that $\lambda(v_i) \in E$ for $i = 1, ..., m$, if $\lambda(v) = \ell$, then $\lambda(v_1)$,..., $\lambda(v_m)$ satisfies $\tau(\ell)$ and the occurrence constraints.
2. for ever node v in $\mathcal{X}$ with children $u_1, ..., u_n$ such that $\lambda(u_i) = @a_i \in A$ for $i = 1, ..., n$, if $\lambda(v) = \ell$, then $\lambda(u_i) = @a_i \in \rho(\ell)$, and $\eta(u_i)$ is a value having datatype $\kappa(@a_i)$.

Now we turn to the mapping language relating a pattern in an XML schema with a formula in an ontology. On the XML side, the basic component is *attribute formulas* [2], which are specified by the syntax $\alpha ::= \ell | \ell(@a_1 = x_1, .., @a_n = x_n)$, where $\ell \in E$, $@a_1, .., @a_n \in A$, E and A are element names and attribute names respectively; and variables $x_1, .., x_n$ are the free variables of α. Tree-pattern formulas over an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ are defined by $\psi ::= \alpha | \alpha[\varphi_1, .., \varphi_n]$, where α ranges over attribute formulas over (E, A). The free variables of a tree formula ψ are the free variables in all the attribute formulas that occur in it. For example, $Company[Department[employee(@eid = x_1)[manager(@mid = x_2)\ [employee(\ @eid = x_3)\ ]]]]$ is a tree formula.

An attribute formula is evaluated in a node of an XML document, and values for free variables come from domain *Dom*. If $\mathcal{X}$ is an XML document over (E, A) and v a node of $\mathcal{X}$, then

- $(\mathcal{X}, v) \models \ell$ iff $\lambda(v) = \ell$, for $\ell \in E$.
- if $\alpha(x_1, ..., x_n) = \ell(@a_1 = x_1, .., @a_n = x_n)$, then $(\mathcal{X}, v) \models \alpha(s_1, ..., s_n)$, where $s_1, ..., s_n \in Dom$, iff $\lambda(v) = \ell$, and for each child v_i of v such that $\lambda(v_i) = @a_i$, $\eta(v_i) = s_i$ for $i \in [1, ..n]$.

Given a document $\mathcal{X}$, a tree-pattern formula $\psi(\overline{x})$, and a tuple $\overline{s}$ from Dom, $\psi(\overline{s})$ is satisfied in $\mathcal{X}$ (written $\mathcal{X} \models \psi(\overline{s})$) if there is a *witness* node v for $\psi(\overline{s})$. Formally, a *witness* node for a $\psi(\overline{s})$ is defined as follows:

- v is a witness node for $\alpha(\overline{s})$, where α is an attribute formula, iff $(\mathcal{X}, v) \models \alpha(\overline{s})$.
- v is witness node for $\alpha(\overline{s})[\ \psi_1(\overline{s}_1),...,\psi_m(\overline{s}_m)]$ iff $(\mathcal{X}, v) \models \alpha(\overline{s})$ and there are m children $v_1, ..., v_m$ of v such that each v_i is a witness node for $\psi_i(\overline{s}_i)$, for $i = 1, ..., m$.

On the ontology side, we use conjunctive formulas with annotations, which treat atomic concepts and roles as unary and binary predicates, respectively. For example, given an ontology containing the atomic concept *Employee* and roles *hasId*, *hasManager*, and *manages*, the following is a mapping formula,
$Company[Department[$
$\quad employee(@eid = x_1)[$
$\quad\quad manager\ (@mid = x_2)\ [$
$\quad\quad\quad employee\ (@eid{=}x_3)\ ]]]] \rightarrow$
$Employee(Y_1)$, $hasId(Y_1, x_1)$, $Employee(Y_2)$, $hasId(Y_2, x_2)$,
$hasManager(Y_1, x_2)$, $Employee(Y_3)$, $hasId(Y_3, x_3)$, $manages(Y_2, Y_3)$::
$identif(Y_1, x_1)$, $identif(Y_2, x_2)$, $identif(Y_3, x_3)$.

There are two sorts of variables. One sort of variables denoted, e.g., by Y_is, represent the individuals in the ontology, and another sort of variables denoted, e.g., by x_js, represent data values containing the attribute values in the XML document and concrete values in the ontology. Since attribute values in the XML document come from the domain Dom, while concrete values in the ontology come from domain $\Delta^{\mathcal{I}}_{\mathbf{D}}$, we assume that each mapping formula implies a set of conversion functions such that when the single variable name x_j is used on both sides, both datatypes in the corresponding positions are matched through an implicit conversion function. We denote by ConstValue the set of all data values that occur in the XML document and we also call them *constant values.* In addition, we assume an infinite set VarValue which we call *variable values* including an infinite set Individual of *individuals* and an infinite set DataValue of *data values*. We require that ConstValue$\cap$VarValue=$\emptyset$.

The annotation comes after :: in the mapping formula. Each predicate in the annotation is of the form $identif(Y, \overline{Z})$ in which Y is an individual variable and $\overline{Z}$ is a tuple of variables. The meaning of $identif(Y, \overline{Z})$ is as follows. The information in XML document indicates that an individual belonging to the concept C in which Y is the placeholder variable, i.e, $C(Y)$ appearing in the formula, can be identified by a set of roles P_1,...,P_n in the ontology, whereas P_1,...,P_n bind Y with $\overline{Z}$ in the formula, i.e., $P_1(Y, Z_1)$,...,$P_n(Y, Z_n)$ appear. We will see later that the annotation is important during the translation and for consistency checking in the ontology. To specify the mapping formulas, we have proposed a semi-automatic tool MAPONTO in [1].

3 The Problem of Translating XML data into Ontologies

We now define the problem of translating XML into ontologies (X-to-O).

Definition 1 (Semantics of Mapping Formulas). *Given an XML schema $\mathcal{S}$ and an ontology $\mathcal{O} =< \mathcal{T}, \mathcal{R}, \mathcal{A} >$, a mapping formula is an expression of the form:*

$$\Psi : \psi(\overline{x}) \rightarrow \varphi(\overline{Y}, \overline{x}) :: annotation. \quad (1)$$

where $\psi(\overline{x})$ is a tree-pattern formula over $\mathcal{S}$, $\varphi(\overline{Y}, \overline{x})$ is a conjunctive formula over atomic concept and role names of $\mathcal{O}$, and $\overline{Y}$ and $\overline{x}$ have no variables in common.

Given an XML document $\mathcal{X}$ conforming to $\mathcal{S}$ and an ontology instance $\mathcal{A}$ consistent with $\mathcal{O}$, we say that the pair $< \mathcal{X}, \mathcal{A} >$ satisfies the formula (1) if whenever there is a tuple $\overline{s}$ such that $\mathcal{X} \models \psi(\overline{s})$, there exists a tuple $\overline{t}$ such that for each assertion β in the formula $\varphi(\overline{t}, \overline{s})$, $\mathcal{A} \models \beta$.

Definition 2 (X-to-O problem). *The problem of translating XML data into ontologies (X-to-O) is a triple $(\mathcal{S}, \mathcal{O}, \Sigma_{\mathcal{SO}})$, where $\mathcal{S} =< E, A, \tau, \rho, \kappa >$ is an XML schema, $\mathcal{O} =< \mathcal{T}, \mathcal{R}, \mathcal{A} >$ is an ontology, and $\Sigma_{\mathcal{SO}}$ is a set of mapping formulas between $\mathcal{S}$ and $\mathcal{O}$.*

Definition 3 (Solutions). *Given an X-to-O problem $\mathcal{P} = (\mathcal{S}, \mathcal{O}, \Sigma_{\mathcal{SO}})$ and an XML document $\mathcal{X}$ conforming to $\mathcal{S}$, an instance $\mathcal{A}$ consistent with $\mathcal{O}$ such that $< \mathcal{X}, \mathcal{A} >$ satisfies all formulas in $\Sigma_{\mathcal{SO}}$ is called a solution for $\mathcal{P}$.*

Recall the *Data Exchange* problem [3, 2]. A data exchange setting is a tuple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, Σ_{st} is a set of *source-to-target dependencies,* or *STDs*, that express the relationship between $\mathbf{S}$ and $\mathbf{T}$, and Σ_t is a set of constraints on the target schema. A solution of the data exchange problem is an instance J over the target schema $\mathbf{T}$ when given an instance I over the source schema $\mathbf{S}$, such that I and J together satisfy all formulas in Σ_{st} and Σ_t. In general, there may be many different solutions for a given instance I, and under target constraints, there may be no solutions at all. If one poses query Q over the target schema, and a source instance I is known, the usual semantics in data exchange uses *certain answers*. A key problem in data exchange is to find a particular solution J_0 so that $\underline{certain}(Q, I)$ can be obtained by evaluating some query over J_0.

Coming back to X-to-O problem, we have defined that the source is an XML schema and the target is an ontology. By analogy, our mapping formulas are the source-to-target dependencies that express the relationships between the XML schema and the ontology. Given an XML document conforming to the XML schema, we want to compute a consistent instance of the ontology, such that the XML document together with the ontology instance satisfy all formulas in the mapping. The major difference from the data exchange problem is that with an ontology as the target, computing a solution calls for a different algorithm.

4 Canonical Solution

As illustrated by the example in Section 1, there could be many solutions for an X-to-O problem. In this section, we define the *canonical solution* in terms of

answering queries against ontologies. For the query language, we use a simple conjunctive query language (CQ_0) which can represent most of the proposed query languages for RDF data (e.g., SPARQL [7]). Formally, a CQ_0 query is of the form:

$$Q : q(\overline{x}) \leftarrow p_1(\overline{Y_1}), p_2(\overline{Y_2}), ..., p_n(\overline{Y_n}). \quad (2)$$

where $\overline{x}$ is a tuple of variables or constants which take values from concrete (datatype) domains (e.g., Integer, String, etc.), $\overline{Y_1}, ..., \overline{Y_n}$ are tuples of variables or constants which take values from both concrete domains and individuals (e.g., object identifiers), and we require $\overline{x} \subset \overline{Y_1} \cup ... \cup \overline{Y_n}$. The predicates $p_1, ..., p_n$ are atomic concept and (abstract and concrete) role names in an ontology. The predicate q is an ordinary predicate with an arity $m = |\overline{x}|$. Let $\underline{\text{Const}}$ denote the set of constants appearing in an ontology, $\underline{\text{Var}}$ denote a set of variables, and $\underline{\text{Const}} \cap \underline{\text{Var}} = \emptyset$. Let $h: \underline{\text{Const}} \cup \underline{\text{Var}} \rightarrow \underline{\text{Const}}$ be a mapping from a tuple of variables or constants to a tuple of constants such that if $c \in \underline{\text{Const}}$, $h(c) = c$. Given an ontology $\mathcal{O} =< \mathcal{T}, \mathcal{R}, \mathcal{A} >$ with an instance $\mathcal{A}$, the answer of the query (2) is defined as a set of tuples of concrete (datatype) values $\{\overline{s}\}$ such that for each tuple $\overline{s}$ there is a mapping h such that $h(\overline{x}) = \overline{s}$ and there are tuples $h(\overline{Y_i})$, $\mathcal{O} \models p_i(h(\overline{Y_i}))$ for each $i = 1, ..., n$. A CQ_0 query only returns tuples consisting of datatype values.

Assume that we are given an X-to-O problem $(\mathcal{S}, \mathcal{O}, \Sigma_{\mathcal{SO}})$, an XML document $\mathcal{X}$ conforming to $\mathcal{S}$, and a CQ_0 query Q against the ontology. What does it mean to answer Q? As in the data exchange problem [3], since there may be many possible solutions to the X-to-O problem, we define the semantics of Q in terms of *certain answers*:

$$\underline{certain}(Q, \mathcal{X}) = \bigcap_{\mathcal{A}' \, is \, a \, solution} Q(\mathcal{A}') \quad (3)$$

where, $Q(\mathcal{A}')$ is the answers of the query Q evaluated over the solution $\mathcal{A}'$. Thus, a tuple $\overline{s}$ of datatype values is in the set of *certain answers* $\underline{certain}(Q, \mathcal{X})$, if $\overline{s} \in Q(\mathcal{A}')$ for every solution $\mathcal{A}'$ of the X-to-O problem.

Definition 4 (Canonical Solution). *Given an X-to-O problem and a CQ_0 query Q against the ontology. A solution $\mathcal{A}$ is a canonical solution if it produces the certain answers when given an XML document $\mathcal{X}$ conforming to the XML schema.*

5 Computing a Canonical Solution

Given an X-to-O problem $\mathcal{P} = (\mathcal{S}, \mathcal{O}, \Sigma_{\mathcal{SO}})$, we assume that the ontology $\mathcal{O} =< \mathcal{T}, \mathcal{R}, \mathcal{A} >$ is satisfiable. For each mapping formula $\Psi : \psi(\overline{x}) \rightarrow \varphi(\overline{Y}, \overline{x}) :: annotation$, the formula $\varphi(\overline{Y}, \overline{x})$ has the form $C_i(Y_i)$,...,P_i (Y_i, Y_j),...,$T_i(Y_i, x_{i_j})$,... where C_i is an atomic concept name, P_i is an abstract role name, T_i is a concrete role name, and $\overline{x}$=$\{...x_{i_j}...\}$. We assume that $\varphi(\overline{Y}, \overline{x})$ is consistent with the ontology $\mathcal{O}$ (written $\mathcal{O} \models \varphi(\overline{Y}, \overline{x})$), which means that for each model $\mathcal{I} =< \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$

of $\mathcal{O}$, the interpretation function $\cdot^{\mathcal{I}}$ can be extended to the variables in $\varphi(\overline{Y}, \overline{x})$ in such a way that $\mathcal{I}$ satisfies every atom in $\varphi(\overline{Y}^{\mathcal{I}}, \overline{x}^{\mathcal{I}})$.

Informally, to compute a *canonical solution* when given a mapping formula Ψ and an XML document $\mathcal{X}$, we start from an initial ontology $\mathcal{O}_0 =< \mathcal{T}, \mathcal{R}, \mathcal{A}_0 >$ and add new assertions $C_i(t_i)$, $P_i(t_i, t_j)$, and $T_i(t_i, s_{i_j})$ in turn to generate a series of ABoxes $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$..., whenever there is a tuple $\overline{s} \in \underline{\text{ConstValue}}$ such that $\mathcal{X} \models \psi(\overline{s})$. The assertions $C_i(t_i)$, $P_i(t_i, t_j)$, and $T_i(t_i, s_{i_j})$ are instantiated from the mapping formula by substituting $\overline{s}$ for $\overline{x}$ and by substituting $\overline{t}$ for $\overline{Y}$, where $\overline{t}$ is a tuple of values in $\underline{\text{Individual}}$. When adding these assertions, some extra assertions will probably be added according to axioms in TBox and RBox. There will be a finite number of ABoxes $\mathcal{A}_1$, $\mathcal{A}_2$, ...,$\mathcal{A}_n$ because there are finite number of tuples in $\mathcal{X}$ satisfying the mapping formula and the propagation we will use terminates. Before presenting the algorithm, we first describe how to generate $\overline{t}$.

Let $\underline{\text{Skolem}}$ be a set of function symbols each of which has an arity. For a function symbol f_C of arity n w.r.t. the concept C, the value of applying f_C to a set of values $d_1, ..., d_n \in \underline{\text{ConstValue}} \cup \underline{\text{Individual}}$ is denoted as $f_C(d_1, .., d_n)$. We require that $f_C(d_1, .., d_n)$ is in the set $\underline{\text{Individual}}$ and two $f_C(d_1, ..., d_n)$ are equal iff they are syntactically equivalent. We choose $\overline{t}$ as follows. Suppose the annotation of the formula Ψ has a predicate $identif(Y_i, \overline{Z})$, where $C_i(Y_i)$ is in Ψ. If $\overline{Z}$ consists of only variables in $\overline{x}$, then let $t_i = f_{C_i}([\overline{Z}/\overline{s}])$ ($[\overline{Z}/\overline{s}]$ means substituting $\overline{s}$ for the variables in $\overline{Z}$ w.r.t. the substitution of $\overline{x}$ in $\varphi(\overline{Y}, \overline{x})$); else $t_i = f_{C_i}([\overline{Z}/\overline{s} \cup \overline{t_j}])$, where each t_j is an individual chosen recursively for individual variables in $\overline{Z}$. The process terminates due to the propagation of the tree structures in XML documents in the annotation.

To detect any inconsistent ABox during the process of computing the canonical solution, we add extra assertions in addition to the assertions instantiated from the mapping formula. A set of propagation rules serves this purpose. The propagation rules are derived from the axioms in the TBox and RBox, and they only apply to the individuals constructed by Skolem functions. We assume that all inclusion axioms in TBox are *concept definition* and the TBox is *acyclic*. That means only axioms of the form $CN \sqsubseteq C$ or $CN \doteq C$ are in TBox, where CN is a concept name, and C does not directly or indirectly refer to CN.

Here are the propagation rules. For an ABox $\mathcal{A}_i$ and individuals t of the form $f_C(\overline{s})$ where f_C is a Skolem function symbol, we add new assertions which do not exist previously to $\mathcal{A}_i$ by the following rules:

1. adding $C(t)$ if $CN(t)$ and $CN \sqsubseteq C$ or $CN \doteq C$ are in TBox;
2. adding $C_1(t)$ and $C_2(t)$ if $(C_1 \sqcap C_2)(t)$ is in $\mathcal{A}_i$;
3. adding $C(t_1)$ ($v \in d$) if $(\forall R.C)(t)$ and $R(t, t_1)$ (resp. $(\forall T.d)(t)$ and $T(t, v)$) are in $\mathcal{A}_i$;
4. adding $R(t, u)$ and $C(u)$ ($T(t, v)$ and $v \in d$) if $(\exists R.C)(t)$ (resp. $(\exists T.d)(t)$) is in $\mathcal{A}_i$ and there is no $R(t, u)$ (resp. $T(t, v)$);
5. adding $P(t, t_1)$ (or $P(t_1, t)$) if $R(t, t_1)$ (or $R(t_1, t)$) is in $\mathcal{A}_i$ and $R \sqsubseteq^* P$ in RBox;
6. replacing t with one of $\{o_1, ..., o_n\}$ if $\{o_1, ..., o_n\}(t)$ is in $\mathcal{A}_i$;

7. adding $C_1(t)$ or $C_2(t)$ if $(C_1 \sqcup C_2)(t)$ is in $\mathcal{A}_i$;
8. replacing the occurrences of t_i with t_j for an individual t_i not computed by a Skolem function and an individual t_j of the form $f_C(\overline{s})$, if $(\leq nR)(t)$ is in $\mathcal{A}_i$ and there exists t_k $k = 1, ..., n+1$ such that $R(t, t_k)$ exists.

Rules 1-5 above are deterministic and rules 6-8 are nondeterministic. There is only one generating rule 4.) which generates u from VarValue but u does not use Skolem functions; therefore, the propagation will terminate for a *tree* structure with depth at most 1.

A *number restriction clash* is the situation in that some abstract role R (resp. concrete role T), $(\leq nR)(t)$ (resp. $(\leq nT)(t)$) is in $\mathcal{A}_i$ and there are $n+1$ different individuals $t_1, ..., t_{n+1}$ (resp. values $v_1, ..., v_{n+1}$) such that $R(t, t_j)$ (resp. $T(t, v_j)$) in $\mathcal{A}_i$ for $j = 1, ..., n+1$. If an ABox $\mathcal{A}_i$ contains a number restriction clash when adding assertions either by instantiating the mapping formula or by applying the propagation rules, then no canonical solution exists and the algorithm returns immediately with an empty solution. Otherwise, the algorithm terminates and generates a series of consistent ABoxes $\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n$. Starting from a consistent initial ABox $\mathcal{A}_0$, for the last ABox $\mathcal{A}_n$, we have the following property:

Lemma 1. *$\mathcal{A}_n$ is a solution.*□

Proof. (Sketch) $\mathcal{A}_n$ contains all assertions which can be constructed from the mapping formulas and tuples $\overline{s}$ such that $\mathcal{X} \models \psi(\overline{s})$. What we need to prove is that $\mathcal{A}_n$ is consistent. To do this, we will use the sound and complete tableaux algorithm for deciding $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases. The algorithm is shown in the papers [4, 5]. They show that if a knowledge base is satisfiable, then the algorithm does not generate any clashes.

The *propagation rules* used in our algorithm for computing a canonical solution is a subset of the *expansion rules* for deciding $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases. However, in contrast to application of the *expansion rules*, our *propagation rules* only apply to individuals computed from the mapping formula and tuples in the XML document. Each individual has the form of $f_C(\overline{s})$.

Then it suffices to prove that if $\mathcal{A}_n$ does not contains the number restriction clash, then the decision procedure does not generate any clashes. □

Further, we have

Proposition 1. *$\mathcal{A}_n$ is a canonical solution. That is, given an XML document $\mathcal{X}$ and a CQ_0 query Q, $Q(\mathcal{A}_n) =$ certain$(Q, \mathcal{X})$.* □

Proof. Suppose Q is $q(\overline{x}) \leftarrow p_1(\overline{Y_1}), ..., p_n(\overline{Y_n})$.

Lemma 1 has shown that $\mathcal{A}_n$ is a solution.

Let $\overline{s}$ be a tuple of data values. If $\overline{s} \in Q(\mathcal{A}_n)$, then there is a mapping h from $\overline{x}$ to $\overline{s}$ such that $(\mathcal{T}, \mathcal{R}, \mathcal{A}_n) \models p_i(h(\overline{Y_i}))$ for each i. We need to prove that for every solution $\mathcal{A}$, $\overline{s} \in Q(\mathcal{A})$, i.e., $(\mathcal{T}, \mathcal{R}, \mathcal{A}) \models p_i(h(\overline{Y_i}))$ for each i. Suppose a mapping formula Ψ is in the form $\psi(\overline{x}) \rightarrow \varphi(\overline{Y}, \overline{x})$:: *annotation*. Let $\mathcal{A}$ be a solution and let $\mathcal{I} = < \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$ be a model of $(\mathcal{T}, \mathcal{R}, \mathcal{A})$. Since $\mathcal{A}$ is a solution w.r.t. $\mathcal{X}$, then $< \mathcal{X}, \mathcal{A} >$ satisfies all Ψs in $\Sigma_{\mathcal{SO}}$, i.e., for a tuple $\overline{s'}$ in $\mathcal{X}$ such

that $\mathcal{X} \models \psi(\overline{s'})$, there is a tuple $\overline{t}$ in Individual such that $\mathcal{A} \models \varphi(\overline{t}, \overline{s'})$ for each formula $\varphi(\overline{Y}, \overline{x})$. By the construction of $\mathcal{A}_n$, we know that if there is a tuple $\overline{s'}$ in $\mathcal{X}$ such that $\mathcal{X} \models \psi(\overline{s}')$, then we add each assertion in the formula $\varphi(\overline{t}, \overline{s'})$ to $\mathcal{A}_n$ and $\mathcal{A}_n$ contains all and only the assertions which can be constructed from all mapping formulas $\varphi(\overline{t}, \overline{s'})$ and the axioms in TBox and RBox. Let $\mathcal{A}'_n$ be the set of assertions instantiated from all formula $\varphi(\overline{t}, \overline{s'})$ and $\mathcal{A}''_n$ be the set of assertions added by applying propagation rules. Since $\mathcal{A} \models \varphi(\overline{t}, \overline{s'})$ for each formula $\varphi(\overline{t}, \overline{s'})$, $\mathcal{A} \models \mathcal{A}'_n$. Hence, the model of $(\mathcal{T}, \mathcal{R}, \mathcal{A})$, $\mathcal{I} =< \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$, is a model of $(\mathcal{T}, \mathcal{R}, \mathcal{A}'_n)$. By the properties of the propagation rules, we know $< \mathcal{T}, \mathcal{R}, \mathcal{A}'_n >\models \mathcal{A}''_n$; therefore, $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{R}, \mathcal{A}_n)$. Since $(\mathcal{T}, \mathcal{R}, \mathcal{A}_n) \models p_i(h(\overline{Y_i}))$ for each i, $\mathcal{I} \models p_i(h(\overline{Y_i}))$ for each i. By this we proved $(\mathcal{T}, \mathcal{R}, \mathcal{A}) \models p_i(h(\overline{Y_i}))$ for each i. Therefore, $\overline{s} \in \underline{certain}(Q, \mathcal{X})$. □

6 Conclusions

We have studied the problem of translating an XML document into an instance of an OWL-DL ontology. However, we are aware that the problem of checking the satisfiability of the ontology and the consistency of the mapping formulas as well as answering conjunctive queries still has a very high complexity – NEXPTIME-complete for OWL-DL ontologies and EXPTIME-complete for OWL Lite ontologies [6, 9]. We attempt to investigate some efficient algorithms for answering conjunctive queries over OWL-DL ontologies, probably incomplete but acceptable, in the future.

Acknowledgments: We are grateful to anonymous reviewers for offering valuable comments, corrections, and suggestions for improvement.

References

1. Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings between XML Data and Ontologies. In ISWC'05, 2005.
2. M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. In PODS'05, Baltimore, USA.
3. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantic and Query Answering. In ICDT'03.
4. I. Horrocks and U. Sattler. Ontology Reasoning in the SHIQ(**D**) Description Logic. In IJCAI'01.
5. I. Horrocks and U. Sattler. A Tableaux Decision Procedure for SHOIQ. In IJCAI'05.
6. I. Horrocks and S. Tessaris. A Conjunctive Query Language for Description Logic Aboxes. In AAAI'00.
7. E. Prud'hommeaux, A. Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/
8. R. Rodriguez-Gianolli and J. Mylopoulos. A Semantic Approach to XML-bases Data Integration. In ER'01.
9. S. Tobies. Complexity Results and Practical Algorithm for Logics in Knowledge Representation. PhD Thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.