A Logical Foundation for XML

Mengchi Liu

School of Computer Science, Carleton University
Ottawa, Ontario, Canada K1S 5B6
mengchi@scs.carleton.ca

Abstract. XML is fast emerging as the dominant standard for data representation and exchange on the World Wide Web. How to view an XML document, i.e., XML data model, and how to query XML documents are two primary research issues for XML. The purpose of this paper is twofold. First, we propose a novel data model for XML that allows us to view XML data in a way similar to complex object data models. Based on this data model, we then investigate how rule-based paradigm can be used to query XML documents and what benefits it brings over existing XML query languages. To this end, we present a rule-based query language called XML-RL, in which we treat existing XML documents as extensional databases, and the queries and functions expressed in rules as intensional databases as in deductive databases. We show that the querying part and result constructing part in XML-RL are strictly separated, the rule body is used to query XML documents and bind variables while the rule head is used to construct the resulting XML document. As several rules can be used for the same query, complex queries can be expressed in XML-RL in a simple and natural way as in logic programming. Also, rules provide a uniform framework for both functions/methods and queries and support recursion in a natural way. Finally, rule-based framework has a formal logic foundation that is lacking in other query languages.

1 Introduction

XML is fast emerging as the dominant standard for data representation and exchange on the World Wide Web. Unlike HTML in which tags are mainly used to describe how to display data, XML tags describe the data itself so that XML data is self-describing. Therefore, a program receiving an XML document can interpret it in multiple ways, can extract, synthesize, and analyze its contents, and restructure it to suit the application's needs.

How to view an XML document, i.e., XML data model, and how to query XML documents are two primary research issues for XML. There are several data models proposed for XML, such as DOM [1], XSL data model [4], XML information set [11], and XML Query Data Model [14]. However, they are low-level data models using nodes and trees and are primarily for machine representation of XML documents.

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 568–583, 2002.

There are also several query languages proposed for extracting and restructuring the XML contents, such as Lorel [3], XML-GL [7], XPath [10], XML-QL [12], XSL [4], XQuery [9], XML Query Algebra [13], etc. Some of them are in the tradition of database query languages, others more closely inspired by XML. The XML Query Working Group has recently published XML Query Requirements for XML query languages [8]. The discussion is going on within the World Wide Web Consortium, within many academic forums and within IT industry, with XQuery been selected as the basis for an official W3C query language for XML. In our view, the main problem with existing query languages, including XQuery, is that the query part and the result constructing part are intermixed, an inherent problem inherited from SQL and OQL [6]. For example, XQuery uses a FOR-LET-WHERE-RETURN (FLWR) construct similar to SQL. The FOR clause plays the same role as the FROM clause in SQL, the LET clause binds a variable to an entire expression, the WHERE clause functions the same as in SQL, and the RETURN clause is analogous to SELECT used to construct/restructure the query results. Like SQL and OQL, simple FLWR construct cannot express complex queries and construct/restructure query results so that the FLWR construct has to be nested in the RETURN clause, and thus makes queries complicated to comprehend.

Also, current work on XML lacks logic foundation to account for various features introduced.

In this paper, we first propose a novel data model for XML that allows us to view XML data in a way similar to complex object data models [2]. Based on this data model, we then investigate how rule-based paradigm can be used to query XML documents and what benefits it brings over existing XML query languages. To this end, we present a rule-based query language called XML-RL, in which we treat existing XML documents as extensional databases, and the queries and functions expressed in rules as intensional databases as in deductive databases. We show that the querying part and result constructing part in XML-RL are strictly separated, the rule body is used to query XML documents and bind variables while the rule head is used to construct the resulting XML document. As several rules can be used for the same query, complex queries can be expressed in XML-RL in a simple and natural way as in logic programming. Also, rules provide a uniform framework for both functions/methods and queries and support recursion in a natural way. Finally, rule-based framework has a formal logic foundation that is lacking in other query languages.

The rest of the paper is organized as follows. Section 2 introduces our data model for XML. Section 3 defines our rule-based query language for XML. Section 4 summarizes and points out further research issues.

2 Data Model for XML

In this section, we introduce our data model for XML. We assume the existence of a set $\mathcal{U}$ of URLs, a set $\mathcal{C}$ of constants, and the symbol *null*.

Definition 1. The notion of *objects* is defined as follows:

(1) null is an *empty* object.
(2) Let $c \in \mathcal{C}$ be a constant. Then c is a *lexical* object.
(3) Let $o_1, ..., o_n$ be objects with $n \geq 0$. Then $\langle o_1, ..., o_n \rangle$ is a *list* object.
(4) Let $a \in \mathcal{C}$ be a constant, and o a lexical object or a list of lexical objects. Then $@a : o$ is an *attribute* object and o is the *value* of the attribute a.
(5) Let $@a_1 : o_1$,...,$@a_m : o_m$ be attribute objects, $p_1, ..., p_n$ be element, lexical, or empty objects with $m \geq 0, n > 0$. Then $o = (@a_1 : o_1, ..., @a_m : o_m, p_1, ..., p_n)$ is a *tuple* object, with two components: attribute component and content component and can be written as $o = @(a_1 : o_1, ..., a_m : o_m)(p_1, ..., p_n)$. When $m = 0$, it is a *pure content* tuple object and can be written as $o = (p_1, ..., p_n)$. When $n = 1$ and p_1 is either *null* or a lexical object, we can also write it as $o = @(a_1 : o_1, ..., a_m : o_m)p_1$.
(6) Let $o = (o_1, ..., o_i, ..., o_n)$ be a tuple object. If o_i is an attribute/element object with a list value, i.e, $o_i = l_i : \langle p_1, ..., p_m \rangle$ with $n \geq 1$, then we can replace it in o with m sequential attribute/element objects as $(o_1, ..., l_i : p_1, ..., l_i : p_m, ..., o_n)$ and vice versa. If every such attribute/element object is replaced in this way, then the resulting tuple object is *flat*. If none of such attribute/element objects are replaced in this way, then o is *normalized*.
(7) Let $e \in \mathcal{C}$ be a constant, and o a tuple object. Then $e : o$ is an *element* object.

The following are examples of objects:

Lexical objects: Computer Science, Database Systems
List objects: ⟨John, Mary⟩, ⟨o123, o234⟩
Attribute objects: @id: o123, @year: 1995, @children: ⟨o123,o234⟩
Tuple object: (@id:o123, @children:⟨o123, o234⟩, title:XML)
Element objects: address:null, title:XML, name:(First:Smith, Last:John)

Lexical objects correspond to character data and attribute values in XML, list objects are used to represent the multiple values of the same attributes or elements, attribute objects to represent attribute-value pairs in XML, tuple objects to represent the relationship among attributes and elements in XML, and element objects to represent element-data pairs, Like other proposals for XML, we use the symbol @ in front of attributes to differ them from elements in tuple objects.

Note that a tuple object in our data model can have several different views: flat view, normalized view and their various mixtures based on Item (6) in Definition 1. The following examples demonstrate their difference:

(@children : o123, @children : o234, title : XML, author : John, author : Tony)
(@children : ⟨o123, o234⟩, title : XML, author : ⟨John, Tony⟩)
(@children : ⟨o123, o234⟩, title : XML, author : John, author : Tony)

The first view is flat as it does not show any list objects. The second view is normalized as it only uses list objects. The last view only shows list objects

for attribute objects but not for element objects and is much closer to XML presentation of data. In the rule-based query language introduced in Section 3, we use the flat view when we have individual variables and use the normalized view when we have list variables.

Conversion between XML and our model is straightforward using the following tranformation operator T:

(1) Let s be a character string or a quoted character string. Then $T(s) = s$.
(2) Let $s = "s_1...s_n"$ be a list of quoted character strings separated by the white space. Then $T(s) = \langle T(s_1), ..., T(s_n)\rangle$.
(3) Let E be a null element: $E = \langle e\ A_1...A_m/\rangle$ or $E = \langle e\ A_1...A_m\rangle\langle/e\rangle$ with $m \geq 0$. Then $T(E) = e : (T(A_1), ..., T(A_m))null$.
(4) Let E be a children or mixed element: $E = \langle e\ A_1...A_m\rangle E_1...E_n\langle/e\rangle$ with $m \geq 0$ and $m > 0$. Then $T(E) = e : (T(A_1), ..., T(A_m), T(E_1), ..., T(E_n))$.

The following are several simple examples:

$T(\text{John}) = \text{John}$ by (1)
$T(\text{"John"}) = \text{John}$ by (1)
$T(\text{"o123 o234"}) = \langle\text{o123,o234}\rangle$ by (2)
$T(\langle\text{person id= "o123"}/\rangle) = \text{person: (@id:o123) null}$ by (3)
$T(\langle\text{name}\rangle\text{John}\langle/\text{name}\rangle) = \text{name: John}$ by (4)
$T(\langle\text{person id="o123"}\rangle\langle\text{name}\rangle\text{John}\langle/\text{name}\rangle\langle/\text{person}\rangle)$
$= \text{person: (@id: o123, name: John)}$ by (4)
$T(\langle\text{person id="o234"}\rangle\text{Tony}\langle/\text{person}\rangle) = \text{person: (@id: o123) Tony}$ by (4)

Example 1. Consider the following XML document:

```
<chapter>
    <title>Data Model</title>
    <section>
        <title> Syntax For Data Model</title>
    </section>
    <section>
        <title>XML</title>
        <section>
            <title>Basic Syntax</title>
        </section>
        <section>
            <title> XML and Semistructured Data</title>
        </section>
    </section>
</chapter>
```

It is transformed into our data model as the following attribute object:

```
chapter: (title: Data Model,
          section: (title: Syntax For Data Model),
          section: (title: XML,
                    section: (title: Basic Syntax),
                    section: (title: XML and Semestructured Data)))
```

Example 2. Consider another example:

```
<bib>
   <book year="1995">
      <title> Introduction to DBMS </title>
      <author> <last> Date </last> <first> C. </first></author>
      <publisher> <name> Addison-Wesley </name > </publisher>
   </book>
   <book year="1998">
      <title> Foundation for ORDB</title>
      <author> <last> Date </last> <first> C. </first></author>
      <author> <last> Darwen </last> <first> D. </first></author>
      <publisher> <name> Addison-Wesley </name > </publisher>
   </book>
   <book></book>
</bib>
```

It is transformed into our data model as follows:

```
bib: (book: (@year:1995,
             title: Introduction to DBMS,
             author: (last: Date, first: C.),
             publisher: (name: Addison-Wesley)),
      book: (@year:1998,
             title: Foundation for ORDB,
             author: (last: Date, first: C.),
             author: (last: Darwen, first: D.),
             publisher: (name: Addison-Wesley))
      book:  null)
```

Note here that the element object is in flat form.

Example 3. Consider the following DTD and the corresponding XML document with ID, IDREF and IDREFS attributes:

```
<!ELEMENT people (person+)>
<!ELEMENT person (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST  person
           id ID #REQUIRD
           mother IDREF #IMPLIED
           children IDREFS #IMPLIED>
```

```
<people>
   <person id="o123">
      <name>Jane</name>
   </person>
   <person id="o234" mother="o456">
      <name>John</name>
   </person>
   <person id="o456" children ="o123 o234">
      <name>Mary</name>
   </person>
</people>
```

It is transformed into our data model as follows:

```
people: (person: (@id:o123, name: Jane),
         person: (@id:o234, @mother: o456, name: John),
         person: (@id:o456, @children: ⟨o123,o234⟩, name: Mary))
```

The following example demonstrate how to represent XML document with mixed content in our data model.

Example 4. Consider the following example:

```
<Address>
  John lives on
  <Street>Main St</Street>
  with house number
  <Number>123</Number>
  in
  <City>Ottawa</City>
  <Country>Canada</Country>
</Address>
```

It is transformed into our data model as follows:

```
Address:(John lives on,
         Street: Main St,
         with house number,
         Number: 123,
         in,
         City: Ottawa,
         Country: Canada)
```

A well formed XML document over the Web has a URL and exactly one root element. We therefore introduce the following notion to represent it in our data model.

Definition 2. Let u be a URL and e be an element object. Then $(u)/e$ is an XML object.

Suppose the XML document shown in Example 3 is found at the URL www.abc.com/people.xml. Then we can represent it as an XML object as follows:

```
(www.abc.com/people.xml)
/people: (person: (@id:o123, name: Jane),
          person: (@id:o234, @mother: o456, name: John),
          person: (@id:o456, @children: ⟨o123,o234⟩, name: Mary))
```

3 Rule-Based Query Language for XML

In this section, we present a rule-based language based on our data model called XML-RL (XML Rule-based Language) to query XML documents. First, we define the syntax. Then we give some examples. Finally, we define the semantics.

3.1 Syntax of XML-RL

Besides the set $\mathcal{U}$ of URLs and the set $\mathcal{C}$ of constants in the data model, XML-RL uses a set $\mathcal{V}$ of variables that is partitioned into two kinds: single-valued variables started with '$' followed by a string and '$' itself is an anonymous variable, and list-valued variables started with '#' followed by a string.

Definition 3. The *terms* are defined recursively as follows:

(1) null is an *empty* term.
(2) Let $c \in \mathcal{C}$ be a constant. Then c is a *lexical* term.
(3) A list value or a list-valued variable is a *list* term.
(4) Let X be a constant or a single-valued variable, and Y a term. Then $@X : Y$ is an *attribute* term, and Y denotes the value of the attribute that X denotes.
(5) Let $X_1, ..., X_n$ be a set of terms with $n \geq 1$. Then $@(X_1, ..., X_n)$ is a *tuple* term.
(6) Let X be a non-anonymous variable and $X_1, ..., X_n, (n \geq 1)$ be a set of attribute terms and element terms. Then $X(X_1, ..., X_n)$ is a *tuple selection* term. When $n = 1$, we can simply use X/X_1 instead.
(7) Let X be a constant or a variable, and Y a term. Then $X : Y$ is an *element* term, and Y denotes the content of the element that X denotes. If Y is an attribute term $X' : Y'$ or a tuple term $(X' : Y')$, then we can simply use $X/X' : Y'$. The case for Y' is the same.
(8) Let X be a single valued variable, Then $\{X\}$ is a *grouping* term.
(9) A single-valued variable is either an *empty* term, *lexical* term, an *attribute* term, an *element* term, or a *tuple* term depending on the context.

The grouping term is used solely to construct query results. The list term is used to manipulate list values. In a tuple selection term $\$t(X_1, ..., X_n)$, $\$t$ is used to denote a tuple and $X_1, ..., X_n$ are used to specify conditions that the tuple $\$t$ must satisfy.

The following are examples of terms:

Lexical terms:	Computer Science, Database Systems, $Name
List terms:	⟨⟩, ⟨John, Mary⟩, ⟨o123, o234⟩, #s
Attribute terms:	@Id:o123, @Id:$x, @year:$y, @$y:1995, @$x:$y
Tuple terms:	(@Id : $x, author : $a), (Title : $t, author : ⟨John⟩), $t
Tuple Selection terms:	$t(publisher : Springer), $t/Title : XML
Element terms:	name:$n, book/author:$a, bib/book: ⟨$b⟩, $x:$y,
Grouping terms:	{title: $t}, {(title : $t, {author : $a})},

A term is *ground* if it has no variables.

Definition 4. The *expressions* are defined as follows:

(1) Let U be a url or a single-valued variable and T an element term. Then $(U)/T$ is a *positive expression*.
(2) If P is a positive expression, then $\neg P$ is a *negative expression*.
(3) Arithmetic, logical, string, and list expressions are defined using terms in the usual way.

The following are several examples of expressions:

Positive exp:	(http://www.abc.com/bib.xml)/people: $p
Negative exp:	¬($url)/people/person: (name: John)
Other exp:	$a = $b × 2, count(#authors) > 2, first(#authors) = John

The positive expression above is equivalent to the following XQuery expression:

```
FOR $p IN document("http://www.abc.com/bib.xml")/people
```

Definition 5. A *rule* has the form $A \Leftarrow L_1, ..., L_n$ and A is the *head* and $L_1, ..., L_n$ is the *body* of the rule, where A is a positive expression, and each L_i is a positive expression, a negative expression, or an arithmetic, logical, string, list expression and only A can contain grouping terms.

The rule body is used to query data in XML documents while the rule head is used to construct the resulting XML document.

We require the variables in the head of the rule to be covered or limited as defined in [5,15,16]. An anonymous variable '$' may appear several times in a rule and their different appearances in general stand for different variables. It cannot appear in the head of a rule.

Definition 6. A *query* is a set of rules.

In order to make queries easier, we adopt XPath abbreviation in rules. That is, if there is a rule: $A \Leftarrow ..., (U)//X, ...$ where // means any number of levels down, then this rule stand for the following rules:

- $A \Leftarrow ..., (U)/X, ...$
- $A \Leftarrow ..., (U)/X_1/X, ...$
- ...

If there are several such Xpath abbreviations in a rule, then it stands for their various combinations as outlined above. Also, if we have $X : ...\$t..., \$t(Y_1, ..., Y_n)$ in the body of a rule, then we can simply use $X : ...\$t(Y_1, ..., Y_n)...$ instead.

If URL U in the head $(U)/T$ is the default one, such as standard output, then we can omit it and use $/T$ instead.

3.2 Query Examples

The following queries are based on the XML documents shown in the last section.

(Q_1) List book elements for books published by Addison-Wesley after 1991.

(file:///home/users/xml/result.xml)
/results/book: $b
⇐
(http://www.abc.com/bib.xml)
/bib/book: $b(publisher/name: Springer, year: $y, title: $t), $y > 1991

Since there are a number (list) of book elements in the XML document, variable $\$b$ matches one book element (a tuple) and tuple selection term specifies the condition that $\$b$ should satisfy. The result is a list of book elements under the root element *results*.

If we simply want to display the result on the screen, then we can simply use the following instead:

/results/book: $b ⇐
(http://www.abc.com/bib.xml)
/bib/book: $b(publisher/name: Springer, year: $y, title: $t), $y > 1991

(Q_2). Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

(file:///home/users/xml/result.xml)
/results/result: (title: $t, author: $a)
⇐
(http://www.abc.com/bib.xml)
/bib/book : (title: $t, author : $a)

Note here the variable $\$a$ in the body matches one author at a time.

(Q_3). For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

(file:///home/users/xml/result.xml)
/results/result: (author: $a, {title: $t})
⇐
(http://www.abc.com/bib.xml)
/bib/book : (title: $t, author: $a)

The grouping term {title: $t} in the head is used to group the titles of all books by author $a.

(Q_4). For each book that has at least two authors, list the title and first two authors.

(file:///home/users/xml/result.xml)
/results/result : (title: $t, {author: $a})
⇐
(http://www.abc.com/bib.xml)
/bib/book : (title: $t, author: #a), count(#a) > 2, $a ∈ first_two(#a)

Note here the variables #a is a list-valued variable that match a list of authors.

(Q_5). For each person, list his/her ancestors IDs.

(file:///home/users/xml/result.xml)
/results/result : (@id: $c, {ancestors: $p})
⇐
(http://www.abc.com/people.xml)
/people/person : (@id: $p, children: #c), $c ∈ #c

(file:///home/users/xml/result.xml)
/results/result : (@id: $c, {ancestors: $p})
⇐
(file:///home/users/xml/result.xml)
/results/result : (@id: $c, ancestors: $a),
(http://www.abc.com/people.xml)
/people/person : (@id: $p, children: #c), $a ∈ #c

The first rule says for each person identified by $p, if $c is a child, then $p is an ancestor of $c. The second rule says if $a is an ancestor of $c, and $a is a child of $p, then $p is also an ancestor of $c. Note here the second rule is recursively defined.

In practice, we can simplify the above query by following Prolog convention to combine the two rules with the same head using ';' as follows:

(file:///home/users/xml/result.xml)
/results/result : (@id: $c, {ancestors: $p})
⇐
(http://www.abc.com/people.xml)
/people/person : (@id: $p, children: #c), $c ∈ #c;
(file:///home/users/xml/result.xml)
/results/result : (@id: $c, ancestors: $a),
(http://www.abc.com/people.xml)
/people/person : (@id: $p, children: #c), $a ∈ #c

3.3 Semantics of XML-RL

In this section, we define the Herbrand-like logical semantics for XML-RL queries.

Definition 7. The *Herbrand universe* U of XML-RL is the set of all ground terms that can be formed.

Definition 8. The *Herbrand base* B of XML-RL is the set of all XML positive expressions that can be formed using terms in U.

Definition 9. An *XML database* XDB is a set of XML objects.

Definition 10. A *ground substitution* θ is a mapping from the set of variables $\mathcal{V} - \{\$\}$ to $U \cup 2^U$. It maps a single-valued variable to an element in U and a list-valued variable to an element in 2^U.

The use of anonymous variables allows us to simply our query rules as demonstrated in the examples above. However, when we deal with semantics, we disallow anonymous variables. We assume that each appearance of anonymous variable is replaced by another variable that never occur in the query rules. This is why we do not map anonymous variable '$' to any object in the above definition.

Given a set of XML documents, our queries just select part of them. Thus, we introduce the following auxiliary notions in order to define the semantics.

Definition 11. A ground term o is *part-of* of another ground term o', denoted by $o \preceq o'$, if and only if one of the following hold:

(1) both are lexical or list objects and $o = o'$;
(2) both are attribute terms or element terms: $o \equiv l : o_i$ and $o' \equiv l : o'_i$ such that $o_i \preceq o'_i$;
(3) both are list terms and $o = o'$;
(4) both are tuple terms: $o \equiv (X_1 : o_1, ..., X_m : o_m)$ and $o' \equiv (X_1 : o'_1, ..., X_n : o'_n)$ with $m \leq n$ such that $o_i \preceq o'_i$ for $1 \leq i \leq m$.

The part-of relationship between o and o' captures the fact that o is part of o'. We need this notion because query results are always based on part of one or more XML documents.

The following are several examples:

$\mathsf{John} \preceq \mathsf{John}$
$\mathsf{author{:}\ (first{:}\ John)} \preceq \mathsf{author{:}\ (first{:}\ John,\ last{:}\ Mary)}$
$\mathsf{(title{:}XML,\ author{:}John)} \preceq \mathsf{(title{:}XML,\ author{:}John,\ author{:}Mary)}$

Definition 12. Let $O = (u)/t$, $O' = (u')/t'$ be two XML objects. Then O is *part-of* O', denoted by $O \preceq O'$, if and only if $u = u'$ and $t \preceq t'$.

Definition 13. Let XDB and XDB' be two XML databases. Then XDB is *part-of* XDB', denoted by $XDB \preceq XDB'$, if and only if for each $O \in XDB - XDB'$, there exists $O' \in XDB' - XDB$ such that $O \preceq O'$.

Definition 14. Let XDB be an XML database. The notion of satisfaction (denoted by $\models$) and its negation (denoted by $\not\models$) based on XDB are defined as follows.

(1) For a ground positive expression $(u)/t$, $XDB \models (u)/t$ if and only if there exists $(u)/t' \in XDB$ such that $t \preceq t'$.
(2) For a ground negative expression $\neg(u)/t$, $XDB \models \neg(u)/t$ if and only if $XDB \not\models (u)/t$
(3) For a ground arithmetic, logical, string, or list expression ψ, $XDB \models \psi$ (or $XDB \models \neg\psi$) if and only if ψ is true (or false) in the usual sense.
(4) For a ground tuple selection term $X(Y_1, ..., Y_n)$. $XDB \models X(Y_1, ..., Y_n)$ if and only if Y_i is satisfied in X for $1 \leq i \leq n$.
(5) For a rule r of the form $A \Leftarrow L_1, ..., L_n$, $XDB \models r$ if and only if for every ground substitution θ, $XDB \models \theta L_1, ..., XDB \models \theta L_n$ implies $XDB \models \theta A$

For example, let XDB denote the XML objects in Example 3. Then we have

$XDB \models \mathsf{(www.abc.com/people.xml)/people/person : (@id : o123, name : Jane)}$
$XDB \models \neg\mathsf{(www.abc.com/people.xml)/people/person : (name : Tony)}$
$XDB \models \mathsf{count}(\langle \mathsf{John}, \mathsf{Mary} \rangle) > 1, \mathsf{first\_one}(\langle \mathsf{John}, \mathsf{Mary} \rangle) = \mathsf{John}, 6 = 3 * 2$
$XDB \models \mathsf{(@id : o123, name : Jane)(name : Jane)}$
$XDB \not\models \mathsf{(@id : o123, name : Jane)(name : Tony)}$

Definition 15. Let Q be a query. A *model* M of Q is a web database that satisfies Q. A model M of Q is *minimal* if and only if for each model N of Q, $M \preceq N$.

Definition 16. Let XDB be an XML database and Q a set of query rules. The *immediate logical consequence* operator T_Q over XDB is defined as follows:

$$T_Q(XDB) = \langle \theta A \mid A \Leftarrow L_1, ..., L_n \in Q \text{ and } \exists \text{ a ground substitution} \theta \text{ such that } XDB \models \theta L_1, ..., XDB \models \theta L_n \}$$

Example 5. Consider query Q_3 in Section 3.2. Then

$T_{Q_3}(DB) =$
⟨(resultURL)/results/result:(author:(last:Date, first: C.),
⟨Title:Introduction to DBMS⟩),
(resultURL)/results/result:(author:(last:Date, first: C.),
⟨Title:Foundation for ORDB⟩),
(resultURL)/results/result:(author:(last:Darwen, first: D.),
⟨Title: Foundation for ORDB ⟩)⟩

where $resultURL \equiv$ *file:///home/users/xml/result.xml.*

Note that the operator T_Q does not perform result construction. Therefore, we introduce the following notions.

Definition 17. Two ground terms o and o' are *compatible* if and only if one of the following holds:

(1) both are constants and are equal;
(2) $o \equiv a : o_i$ and $o' \equiv a : o'_i$ such that o_i and o'_i are compatible;
(3) both are grouping terms;
(4) $o \equiv (X_1, ..., X_n)$ and $o' \equiv (X'_1, ..., X'_n)$ such that X_i and X'_i are compatible for $1 \leq i \leq m$.

For example, the following pairs are compatible:

John and John
Author:(last:Date, first: C.) and Author:(last:Date, first: C.)
⟨Title : Databases⟩ and ⟨Title : XML⟩
(author:(last:Date, first: C.), ⟨Title: Database⟩) and
(author:(last:Date, first: C.), ⟨Title: Foundation for ORDB⟩)

Definition 18. Two ground positive expression $(u)/o$ and $(u')/o'$ are *compatible* if and only if $u = u'$ and o and o' are compatible. A set of ground positive expressions are *compatible* if and only if each pair of them are compatible.

For example, the set of ground positive expressions in Example 5 are compatible.

Definition 19. Let S be a set of ground positive expressions (terms) and S' a compatible subset of S. Then S' is a *maximal* compatible set in S if there does not exist a ground positive expressions (terms) $o \in S - S'$ that is compatible with each element in S'.

Definition 20. Let S be a list of ground terms and $\uplus$ list concatenation operator. Then the *constructing* operator C is defined recursively on S as follows:

(1) If S is partitioned into n maximal compatible sets: $S = S_1 \uplus ... \uplus S_n$ with $n > 1$, then $C(S) = \langle C(S_1), ..., C(S_n) \rangle$.

(2) $S = \langle o \rangle$ then $C(S) = o$.
(3) If S is a compatible set of attribute terms or element terms: $S = \langle a : o_1, ..., a : o_n \rangle$, then $C(S) = a : C(\langle o_1, ..., o_n \rangle)$.
(4) Let $S = \langle \langle X_1 \rangle, ..., \langle X_n \rangle \rangle$ be a set of grouping terms of attribute or element objects. If $n = 1$, then $C(S) = C(X_1)$. Otherwise, $C(S) = (C(X_1), ..., C(X_n))$.
(5) Let S be a set of compatible tuples $S = \langle (X_1, ..., X_m, Y_1), ...(X_1, ..., X_m, Y_n) \rangle$ where $X_1, ..., X_m$ are non-grouping terms and $Y_1, ..., Y_n$ are grouping terms. Then $C(S) = (C(X_1), ..., C(X_m), C(Y_1), ..., C(Y_n))$.

The following is an constructing example:

$C(\langle$bib: $\langle$book: (Title: Web)$\rangle$,
bib: $\langle$book: (Title: Databases)$\rangle$,
bib: $\langle$Journal: (Title: XML)$\rangle\rangle)$
= bib : $C(\langle\langle$book: (Title: Web)$\rangle$,
$\langle$book: (Title: Databases)$\rangle$,
$\langle$Journal: (Title: XML)$\rangle\rangle)$ by (3)
= bib : (book: (Title: Web),
book: (Title: Databases),
Journal: (Title: XML)) by (4)

We extend the constructing operator to a list of ground positive expressions as follows: if S is a compatible list of objects of the form $(u)/o_1, ..., (u)/o_n$, then $C(S) = (u)/C(\langle o_1, ..., o_n \rangle)$. Otherwise, C is not defined.

Definition 21. The *powers* of the operation T_Q over the XML database XDB are defined as follows:

$T_Q \uparrow 0(DB) = DB$
$T_Q \uparrow n(DB) = T_Q(C(T_Q \uparrow n-1(DB))) \cup T_Q \uparrow n-1(DB)$ if C is defined
$T_Q \uparrow \omega(DB) = \cup_{n=0}^{\infty} T_Q \uparrow n(DB) - DB$ if $T_Q \uparrow n(DB)$ is defined

Continue with Example 5, we have

$C(T_{Q_3} \uparrow \omega(DB))$
= $C(\langle$(resultURL)/results/result :(author : (last : Date, first : C.),
$\langle$Title : IntroductiontoDBMS$\rangle$),
(resultURL)/results/result : (author : (last : Date, first : C.),
$\langle$Title : FoundationforORDB$\rangle$),
(resultURL)/results/result : (author : (last : Darwen, first : D.),
$\langle$Title : FoundationforORDB$\rangle\rangle)$
=(resultURL)/results : $C(\langle$result : (author :(last : Date, first : C.),
$\langle$Title : IntroductiontoDBMS$\rangle$),
result :(author :(last : Date, first : C.),
$\langle$Title : FoundationforORDB$\rangle$),
result :(author :(last : Darwen, first : D.),
$\langle$Title : FoundationforORDB$\rangle\rangle)$ by (3)

$$
\begin{aligned}
=&(\mathsf{resultURL})/\mathsf{results} : (C(\langle \mathsf{result}{:}(\ \mathsf{author} :(\mathsf{last{:}Date},\ \mathsf{first{:}\ C.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{IntroductiontoDBMS}\rangle)),\\
&\qquad\qquad\qquad\qquad\quad \mathsf{result} : (\mathsf{author} :(\mathsf{last{:}Date},\ \mathsf{first{:}\ C.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{Foundation\ for\ ORDB}\rangle)\rangle),\\
&\qquad\qquad\qquad\qquad C(\langle \mathsf{result} : (\mathsf{author} :(\mathsf{last{:}Darwen},\ \mathsf{first{:}\ D.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{Foundation\ for\ ORDB}\rangle)\rangle)) \quad \text{by (1)}\\
=&(\mathsf{resultURL})/\mathsf{results}{:}(\mathsf{result} : C(\langle(\mathsf{author} :(\mathsf{last{:}Date},\ \mathsf{first{:}\ C.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{Introduction\ to\ DBMS}\rangle)),\\
&\qquad\qquad\qquad\qquad\qquad\quad (\mathsf{author} :(\mathsf{last{:}Date},\ \mathsf{first{:}\ C.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{Foundation\ for\ ORDB}\rangle)\rangle),\\
&\qquad\qquad\qquad\quad \mathsf{result} : C(\langle(\mathsf{author} :(\mathsf{last{:}Darwen},\ \mathsf{first{:}\ D.}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \langle \mathsf{Title} : \mathsf{Foundation\ for\ ORDB}\rangle)\rangle)) \quad \text{by (3)}\\
=&(\mathsf{resultURL})/\mathsf{results} : [\mathsf{result} : (\mathsf{author} :(\mathsf{last{:}Date},\ \mathsf{first{:}\ C.}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{Title} :\ \mathsf{Introduction\ to\ DBMS},\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{Title} :\ \mathsf{Foundation\ for\ ORDB}\rangle),\\
&\qquad\qquad\qquad\quad \mathsf{result} : (\mathsf{author} :(\mathsf{last{:}Darwen},\ \mathsf{first{:}\ D.}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{Title} :\ \mathsf{Foundation\ for\ ORDB})) \qquad\qquad \text{by (5)}
\end{aligned}
$$

Theorem 1. Let XDB be a web database and Q a set of query rules. Then $C(T_Q \uparrow \omega(XDB))$ is a minimal model of Q.

Definition 22. Let XDB be an XML database and Q a set of query rules. Then the *semantics* of Q under XDB is given by $C(T_Q \uparrow \omega(XDB))$.

4 Conclusion

In this paper, we have proposed a novel data model for XML that is able to represent XML documents in a natural and direct way. It establishes relationship between XML and complex object data models. Using this data model, we can easily comprehend XML from database point of view.

We have also presented a rule-based query language based on the data model proposed. Formal syntax and logic-based declarative semantics of XML-RL are presented. XML-RL provides a simple but very powerful way to query XML documents and to construct/restructure the query results. As defined, XML-RL is capable of handling (recursive) queries that make use of partial information of the given XML documents. A number of XML-RL queries are included in the paper to demonstrate the usefulness of XML-RL.

The main novel features that make XML-RL simple to use is the introduction of grouping terms and the corresponding constructing operator so that the querying part and the result constructing part can be strictly separated. More importantly, we have provided a formal logic foundation that is lacking in other XML query languages.

Indeed, other kinds of database query languages, such as calculus-based, can also be developed based on our data model and the work done in the database community for nested-relational and complex object databases.

We have implemented XML-RL using Java. The system will soon be available from the Web site at `http://www.scs.carleton.ca/~mengchi/XML-RL/` after further testing and debugging.

The language presented here is not a full-fledge one as our primary focus is on the formal foundation. However, many other features can be added. We leave it as our future work.

References

1. Document Object Model (DOM). *http://www.w3.org/DOM/*. 568
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995. 569
3. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *Intl. Journal of Digital Libraries*, 1(1):68–88, 1997. 569
4. S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, and S. Zillies. Extensible Stylesheet Language (XSL) Version 1.0. *http://www.w3.org/TR/2000/CR-xsl-20001121*, November 2001. 568, 569
5. C. Beeri, S. Naqvi, O. Shmueli, and S. Tsur. Set Construction in a Logic Database Language. *Journal of Logic Programming*, 10(3,4):181–232, 1991. 575
6. R. G. G. Cattell and D. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, Los Altos, CA, 1997. 569
7. S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a Graphical Language for Querying and Restructuring WWW data. In *Proceedings of the 8th International World Wide Web Conference*, Toronto, Canada, 1999. 569
8. D. Chamberlin, P. Fankhauser, M. Marchiori, and J. Robie. XML Query Requirements. *http://www.w3.org/TR/2001/WD-xmlquery-req-20010215*, February 2001. 569
9. D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery: A Query Languge for XML. *http://www.w3.org/TR/2001/WD-xquery-20010215*, February 2001. 569
10. J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. *http://www.w3.org/TR/1999/REC-xpath-19991116*, November 2001. 569
11. J. Cowan and R. Tobin. XML Information Set Data Model. *http://www.w3.org/TR/xml-infoset*, May 2001. 568
12. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. *http://www.w3.org/TR/1998/Note-xml-ql-19980819*, August 1998. 569
13. P. Fankhauser, M. Fernández, A. Malhotra, M. Rys, J. Siméon, and P. Wadler. The XML Query Algebra. *http://www.w3.org/TR/2001/WD-Query-algebra-20010215*, February 2001. 569
14. M. Fernandez and J. Robie. XML Query Data Model. *http://www.w3.org/TR/2001/WD-Query-datamodel-20010215*, February 2001. 568
15. M. Liu. Relationlog: A Typed Extension to Datalog with Sets and Tuples. *Journal of Logic Programming*, 36(3):271–299, 1998. 575
16. J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988. 575