XDO2: A Deductive Object-Oriented Query Language for XML

Wei Zhang[1], Tok Wang Ling[1], Zhuo Chen[1], and Gillian Dobbie[2]

[1]School of Computing, National University of Singapore,
Lower Kent Ridge Road, Singapore 119260
{zhangwe2, lingtw, chenzhuo}@comp.nus.edu.sg
[2]Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
gill@cs.auckland.ac.nz

Abstract. In the past decade, researchers have combined deductive and object-oriented features to produce systems that are powerful and have excellent modeling capabilities. More recently, an XML query language XTree was proposed. Queries written in XTree are more compact, more convenient to write and easier to understand than queries written in XPath. In this paper, we introduce a novel XML query language XDO2 that extends *XTree*, with *deductive* features such as *deductive rules* and *negation*, and *object-oriented* features such as *inheritance* and *methods*. Our XDO2 language is more *compact*, and *convenient* to use than current query languages for XML such as XQuery and XPath because it is based on XTree, supports (recursive) deductive rules and the not-predicate. An XDO2 database example is given to motivate the usefulness of the language. The formal treatment of language syntax and semantics are presented in the appendices.

Keywords: XML query language, deductive rule, not-predicate negation, fixpoint semantics

1 Introduction

In the past decade, a large number of deductive object-oriented database systems have been proposed, such as F-logic [6], ROL [12] and DO2 [10]. Based on these proposals and other work in the area of object-oriented data models, such as O2 [4] and Orion [7], a large number of deductive and object-oriented features have been investigated. The two most important features are deductive rules and inheritance.

XML is fast emerging as the dominant standard for data representation and exchange on the web. Many query languages have been proposed in the past few years, such as XPath [3], XQuery [1], declarative XML query languages such as [14] and XTree [2]. Although XPath has been widely adopted for XML querying, XTree has been recently proposed as a declarative XML query language which is more compact, more convenient to write and understand than XPath. However,

to the best of our knowledge, there is no XML query language that can support deductive rules and object-oriented features in the XML querying community. In this paper, we introduce a novel XML query language XDO2 which extends XTree with deductive database features such as *deductive rules* and *negation*, and object-oriented features such as *inheritance* and *methods*.

In this paper, we present the language XDO2, highlighting its salient features. We present the full syntax and semantics of the language in the appendices.

The major contributions of the XDO2 query language are:

1. Negation is supported in the XDO2 language with semantics similar to the not-predicate [8] instead of the conventional logical negation symbol "$\sim$" which is used in XQuery. A consequence of this decision is that XDO2 is able to support nested negation and negation of sub-trees.
2. Methods that deduce new properties are implemented as deductive rules. XDO2 can use the new properties directly. The presence of recursive deductive rules makes recursive querying possible.
3. Schema querying is made possible with a special term *stru : value* to explicitly distinguish the element name from the element value (or element content). *stru* binds to the element name and *value* binds to the element value. Unlike in XQuery, the name and value pair are bound to the variables together.
4. Inheritance enables a subclass object to inherit all the attributes and subelements from its superclass objects. These inherited properties can be directly used in querying.
5. Features such as the binding of multiple variables in one expression, compact return format and explicit multi-valued variables are supported in the XDO2 language naturally due to the influence of XTree [2].

The rest of this paper is organized as follows. We provide a brief introduction to XTree in section 2. We introduce an XDO2 database example in section 3. Section 4 presents and discusses the most salient features of the XDO2 query language. Section 5 compares our language with related languages. Section 6 summarizes this paper and points out some future research directions. The syntax and semantics of the XDO2 language are presented in the appendices.

2 Background

XTree [2] was recently proposed as an alternative to the XML query language XPath. The main advantages of XTree over XPath are:

1. XPath describes a linear path to the target XML node set. In the querying part of a query, one XPath expression can only bind one variable. However, XTree has a tree structure which is similar to the structure of an XML document. In the querying part of a query, one XTree expression can bind multiple variables.

2. XPath cannot be used to define the return format. However, in the result construction part of a query, one XTree expression can be used to define the result format. This effectively avoids the nested structure in the query.
3. XPath does not express multi-valued variables explicitly. However, in XTree expressions, multi-valued variables are explicitly indicated, and their values are uniquely determined. Some natural built-in functions are defined to manipulate multi-valued variables in an object-oriented fashion.

Thus, although XPath and XTree have the same expressive power (i.e., any query that can be expressed by XTree can also be expressed by several XPath expressions), XTree is more compact and convenient to use than XPath, and queries based on XTree expressions are shorter in length and easier to write and comprehend. In short, XTree takes the XML tree structure into consideration while XPath does not. For more details, please refer to XTree [2].

3 An XDO2 Database Example

In this section, using an example, we demonstrate many of the features of the XDO2 language. We show an XDO2 database, *Person_Company_Employee*, which combines features from XML, deductive databases, and object-oriented databases. Section 3.1 presents the database schema, and explains how to express deductive rules and inheritance in the database. Section 3.2 presents the XML database data, including the extensional data from XML data element facts, intensional data from deductive rules, and the class hierarchy relationships. An XDO2 query with its result is also presented. The syntax and semantics of the XDO2 language are presented in the appendices.

3.1 Schema and Rules

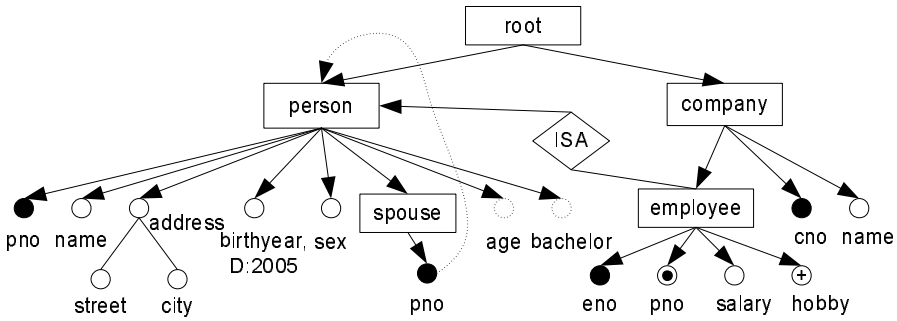

Fig. 1. Person_Company_Employee ORASS schema diagram

The ORA-SS schema model [9] is used to represent the schema, with extensions to model the deductive and inheritance features in Figure 1. In the schema diagram, there are four object classes *person*, *company*, *employee* and *spouse* represented as rectangles. In the *person* class, we model the *age* as a derived attribute indicated by dashed circles. This is because *person* has an attribute *birthyear*, and *age* can be derived using *birthyear*. Another derived attribute is *bachelor* which can be derived using the person's *sex* and *spouse*. The identifier of *person* is *pno*. The *employee* class is a subclass of the *person* class, and inherits all the attributes and derived attributes from the *person* class. The inheritance relationship is denoted by the *ISA* diamond in the schema diagram. The identifier of *employee* is *eno*. The candidate identifier *pno* indicated by a filled circle inside a circle in *employee* class is from the *person* object class.

In this example, we can see the two new features that are not present in XML databases: *derived attribute* and *class inheritance*. Class inheritance is supported in XML schema [5]. We now highlight how to define the derived attributes of object classes. In object-oriented programming languages, methods are defined using functions or procedures and are encapsulated in class definitions. In deductive databases, rules are used instead of functions and procedures. By analogy, derived attributes or methods in XDO2 are defined using deductive rules and encapsulated in class definitions.

In the following, we use a deductive rule to define the method *age* encapsulated in object class *person*.

$\$p/age : \$a$:- $/root/person : \$p/birthyear : \$b, \$a = 2005 - \$b$.

This rule says if there is a *person* element under the *root* element, and the *person* has sub-element *birthyear*, then the age is equal to 2004 minus the birthyear. In the method *age* above, the notation ":-" means if a substitution of all variables to values makes the right hand side true, then the left hand side is also true. In the method, there are predicates $\$p/age : \$a$, $/root/person : \$p/birthyear : \$b$, and $\$a = 2004 - \$b$. The notation ":" binds the value of the left hand side to the right hand side. If the left hand side is an object class, then the right hand side binds to the object identifier, such as $p binds to the person's identifier. Otherwise, it binds to the value of the left hand side. The single-valued variable is denoted by a "$" followed by a string literal.

The below rule defines the method *bachelor* encapsulated in object class *person*.

$\$p/bachelor : true$:- $/root/person : \$p/[sex : "Male", not(spouse : \$s)]$.

This rule says if a *person* element under *root* element has an attribute *sex* with string value *"Male"*, and this same person does not have a *spouse*, then the derived attribute *bachelor* of the object class *person* has boolean value *true*. The two boolean values *true* and *false* are reserved in the language. The notation "[ ]" in the *bachelor* method above is used to group the attributes, elements or methods which are directly defined under the same parent element, such as *person* in this case. The notation "not"[8] negates the predicate expression.

```
<root>
   <person pno="p1">
      <name>John</name>
      <address>
         <street>King</street>
         <city>Ottawa</city>
      </address>
      <birthyear>1975</birthyear>
      <sex>Male</sex>
   </person>
   <person pno="p2">
      <spouse pno="p3" />
      <name>Mike</name>
      <address>
         <street>Albert</street>
         <city>Ottawa</city>
      </address>
      <birthyear>1954</birthyear>
      <sex>Male</sex>
   </person>
   <person pno="p3">
      <spouse pno="p2" />
      <name>Mary</name>
      <address>
         <street>Albert</street>
         <city>Ottawa</city>
      </address>
      <birthyear>1958</birthyear>
      <sex>Female</sex>
   </person>
   <company cno="c1">
      <name>Star</name>
      <employee eno="e1" pno="p1">
         <salary>6000</salary>
         <hobby>Tennis</hobby>
         <hobby>Soccer</hobby>
      </employee>
      <employee eno="e2" pno="p2">
         <salary>4000</salary>
         <hobby>Tennis</hobby>
      </employee>
   </company>
</root>
```

(a) XML extensional database

```
% Rule R1 defines that the age of a
% person is 2005 minus his/her
% birthyear.

(R1)  $p/age : $a :- /root/person : $p/
         birthyear : $b, $a = 2005 - $b.

% Rule R2 defines that a person is a
% bachelor if he is a male and without
% spouse.

(R2)  $p/bachelor : true :- /root/
         person : $p/[sex : "Male",
         not(spouse : $s)].
```

(b) XML intensional database

```
employee ISA person
by employee.pno ISA person.pno
```

(c) XML class hierarchy relationships

Fig. 2. Person_Company_Employee database

3.2 Data and Query

The data or instance of the *Person_Company_Employee* database is shown in Figure 2. There are three parts to the database: the *XML extensional database*, the *XML intensional database*, and the *XML class hierarchy relationships*. The XML extensional database contains the XML data element facts with their tree structure. The XML intensional database contains the deductive rules which can be used to derive new XML data elements or attributes from the extensional database. The XML class hierarchy relationships define the object class hierarchy in the database such as *employee* is a subclass of *person*. Multiple inheritances are allowed and we can resolve the multiple inheritance conflicts using the explicit selection technique adopted from [11]. Storing the deductive rules and class hierarchy relationships in the XML database system, enables querying using deductive rules and the class hierarchy, as shown in the following example.

Example 1. This query retrieves the age and salary of all employees who are bachelors, with *age* less than 30, and *salary* larger than 5000.

$$/db/youngRichBachelor : \$e/[age : \$a, payroll : \$s] \Leftarrow /root/company/ \\ employee : \$e/[age : \$a, bachelor : true, salary : \$s], \$a < 30, \$s > 5000.$$

Notice the query format is similar to the deductive rule used to describe methods. The notation "$\Leftarrow$" separates the return format of the query from the query and conditional part. The left hand side is used to define the XML result format, like in the *return* clause in XQuery, and the right hand side is the query and the conditional parts like the *for*, *let* and *where* clauses in XQuery. Therefore, our XDO2 query language is more simple and compact with only one line of some predicate expressions instead of the FLWR clause in XQuery. With the deductive rules and the inheritance feature defined in the XML database, the user can directly query the attributes or methods both in *employee* and its superclass *person*, such as *age* and *bachelor* in the example.

Using the XDO2 database in Figure 2, only employee *'e1'*, whose *pno* is *'p1'* satisfies the conditions. The *youngRichBachelor* element and its two subelements *age* and *payroll* form the query result as follows.

```
<db>
   <youngRichBachelor eno="e1">
      <age>29</age>
      <payroll>6000</payroll>
   </youngRichBachelor>
</db>
```

Notice $e binds to the object identifier value of the employee object, i.e., eno value. The attributes of *youngRichBachelor* element, *age* and *payroll* are from the derived attribute *age* of *person* object *'p1'* inherited by *'e1'*, and *salary* of *employee* object *'e1'* respectively.

4 XDO2 Language Features

In the example of section 3, we have shown how to use deductive rules to define methods so that the query language can be simplified greatly and made more compact. It also shows the advantages of the inheritance feature. In this section, we will present some other important features of the XDO2 query language. Specifically, they are multi-valued variables and aggregat function, schema querying, negation using the not-predicate [8], and querying using recursive deductive rules.

4.1 Multi-valued Variables

We use the expressions *<$var>* and *{$var}* to represent list-valued variables and set-valued variables respectively. Functions that are defined on lists and sets are consistently expressed in an object-oriented fashion.

Example 2. Consider the following query that returns the titles of the books that have more than one author written in XDO2.

XDO2 query expression:

/result/multiAuthorBook/title : $t ⇐
/bib/book/[title:$t, author:<$a>], <$a>.count()>1.

XQuery expression:

for $book *in* */bib/book*, $t *in* $book/title
let $a *in* $book/author
where count($a) > 1
return <result><multiAuthorBook>{$t}</multiAuthorBook></result>

In the XDO2 query, the variable *<$a>* is bound to the list of authors for each book, and the variable $t is bound to the title of the book. The square brackets [ ] enclosing *title* and *author* specifies that these two elements are siblings, and share a common parent.

4.2 Schema Querying

We use the term *stru : value* to explicitly distinguish the element (or attribute) name from the value of the element (or attribute). It provides simple and natural facilities for exploring the structure or schema of the XML data. The user can put the variable such as $v in the left side of the : symbol to bind to the attribute name or element tag. Unlike in XQuery, both the structure and value can be bound to a variable.

Example 3. Consider the following query that finds two sibling element tags with value "King" and "Ottawa" directly or indirectly under person.

⇐ */root/person//*[$*ele*1 : "*King*", $*ele*2 : "*Ottawa*"].

In this query, we omit the query result format. The square brackets enclosing $ele1 and $ele2 specifies that these two elements are siblings. The path abbreviation "//" is used to indicate they are directly or indirectly under person. The two variables $ele1 and $ele2 are used to bind the element tags that have the values as specified. Using the data from Figure 2, $ele1 = street and $ele2 = city satisfy the query.

4.3 Negation Querying

In deductive databases, negation makes the rules more powerful and queries more meaningful. However, it complicates the query's interpretation and evaluation. To represent negation in XDO2, we chose the not-predicate [8] instead of the conventional logical negation symbol "∼" to express negation. It has been noted in [8] that the not-predicate is not always equivalent to "∼" in negation expressions. The main difference between the not-predicate and "∼" lies in the interpretation of the uninstantiable variables (i.e. variables that do not appear in any positive expression in the body of the rule or query) in the negation expression. Otherwise, they are equivalent. Using the not-predicate, the uninstantiable variables are existentially quantified while they are universally quantified using "∼". For the justification, please refer to [16].

As we know, XQuery [1] provides a function *not()* which needs a boolean value as its argument and is similar to "∼", and it does not support the not-predicate operator. The function *not()* is usually combined with *some* and *every* quantifiers. However, by using the not-predicate operator alone in XDO2, we can achieve the same expressive power and make our queries more simple and compact. In addition, the function *not()* in XQuery can only be applied to one XPath expression, but not to a sub-tree structure. However since we have tree-structure expressions in the XDO2 language, we can express the sub-tree structures naturally. Two examples are shown as follows.

Example 4. Consider the following query expressed in XDO2 and XQuery that retrieves the company name of companies where each employee of the company has hobby "Tennis".

XDO2 query expression:

$/db/allLikeTennisCom : \$n \Leftarrow /root/company : \$c/name : \$n,$
$\$c/not(employee/not(hobby : "Tennis")).$

XQuery expression:

```
for $c in /root/company
where EVERY $e IN $c/employee SATISFIES
      SOME $h IN $e/hobby SATISFIES string($h)="Tennis"
return <db><allLikeTennisCom>{string($c/name)}
                  </allLikeTennisCom></db>
```

Example 5. Consider the following query that retrieves the companies which do not have employees who have sex "Male" and birthyear 1975.

XDO2 query expression:

$/db/company : \$c \Leftarrow /root/company : \$c/not(employee/$
$[sex : \text{“}Male\text{”}, birthyear : 1975]).$

XQuery expression:

for $c in /root/company
where NOT (SOME $e IN $c/employee SATISFIES
($e/sex = “Male” AND $e/birthyear = 1975))
return <db>{$c}</db>

As we can see from the two examples, our XDO2 query using the not-predicate is much more simple and compact compared with the XQuery expression which needs the key word “EVERY”, “SOME”, “NOT”, “IN”, “SATISFIES”, “AND” to express the same meaning.

4.4 Recursion Querying

In deductive databases, it is natural to define a recursive query using recursive deductive rules. Similarly, in XDO2, we also support recursive deductive rules and make the recursive query possible to extend the expressive power of the XDO2 language.

Example 6. Suppose there are child sub-elements directly under the person element. The following deductive rules define descendants of a person.

(R3) $\$p/descendant : \$c$:- $/root/person : \$p/child : \$c.$

(R4) $\$p/descendant : \$d$:- $/root/person : \$p/child : \$c,$
$\$c/descendant : \$d.$

The rule R4 says for each person bound to $p, if $c is his/her child, then $c is a descendant of $p. The rule R5 says if $c is a child of $p, and $d is a descendant of $c, then $d is also a descendant of $p. Note the rule R5 is recursively defined. Using the rules defined, we can write a recursive query to retrieve all the descendants of a person with identifier (i.e. pno) value ‘p1’ as follows,

$\Leftarrow /root/person : \text{‘}p1\text{’}/descendant : \$d.$

5 Comparison with Related Work

The success of *F-logic* [6] was due to the clean combination of the object-oriented and deductive paradigms. *Flora-2* [15] extended F-logic for the semantic web. However, the underlying data in F-logic and Flora-2 are objects and can not handle the current popular XML tree data structure. The XDO2 language is designed for the XML tree data while including the deductive and object-oriented features. Many languages, such as *XPath* [3], *XQuery* [1], and *XTree* [2] have been proposed for querying XML documents. However, they can not support (recursive) deductive rules which can be used to derive new properties to simplify the querying as in XDO2. The *XML_RL* [13] for XML is a language with

the deductive features, however this query languages does not support object-oriented inheritance. Furthermore, since XDO2 is based on XTree, where queries are more compact, more convenient to write and understand than XPath queries, the XDO2 inherits these merits. Another major difference between XDO2 and other logical query languages for XML lies in the use of the not-predicate [8] for querying. As section 4.2 shows, the XDO2 query using the not-predicate is much more simple and compact compared with the XQuery expressions.

	XDO2	**XQuery**	**XTree Query**	**F-logic**	**XML_RL**
Underlying data	XML tree	XML tree	XML tree	Object	XML tree
Path expression	XTree	XPath	XTree	Path expression	XTree-like expression
Deductive rule	Yes	No	No	Yes	Partial
Recursion	recursive rules	recursive function	recursive query	recursive rule	recursive query
Negation	not-predicate	logical negation	logical negation	logical negation	logical negation
Quantification	No need	Yes	Yes	Yes	Yes
Multi-valued variable	Yes	No	Yes	No	Yes
Direct structure querying	Yes	No	Yes	Yes	Yes
Object-oriented features	Yes	No	No	Yes	No

Table 1. Comparison between XML query languages

A summary of the comparison with other XML query languages is shown in table 1.

6 Conclusion

Deductive databases and object-oriented databases are two extensions of the current relational database systems. Guided by this, we propose a novel new XML query language XDO2 with deductive database features such as *deductive rules* and *negation*, and object-oriented features such as *inheritance* and *methods*. Our XDO2 language is more *compact*, and *convenient* to use than current query languages for XML such as XQuery, XPath and XML_RL[13] because it is based on XTree [2], supports (recursive) deductive rules, not-predicate negation and schema querying. An XDO2 database example is presented to motivate the usefulness of the language. In the appendices, we present a formal treatment of the XDO2 language syntax and semantics.

In the future we would like to investigate how to evaluate the queries efficiently, especially for the not-predicate and recursive queries.

References

1. D. Chamberlin, D. Florescu, J. Robie, J. Simon, and M. Stefanescu. XQuery 1.0: A query language for XML, May 2003. http://www.w3.org/TR/xquery.
2. Zhuo Chen, Tok Wang Ling, Mengchi Liu, and Gillian Dobbie. XTree for declarative XML querying. In *Proceedings of DASFAA*, pages 100–112, Korea, 2004.
3. J. Clark and S. DeRose. XML path language(XPath) version 1.0, November 2001. http://www.w3.org/TR/xpath.
4. O. Deux et al. The story of O2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, 1990.
5. D.C. Fallside. XML schema part 0: Primer, May 2001. http://www.w3.org/TR/xmlschema-0.
6. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42(4):741–843, 1995.
7. W. Kim. *Introduction to object-oriented databases*. The MIT Press, Cambridge Massachusetts, 1990.
8. Tok Wang Ling. The prolog not-predicate and negation as failure rule. *New Generation Computing*, 8(1):5–31, 1990.
9. Tok Wang Ling, Mong Li Lee, and Gillian Dobbie. *Semistructured Database Design*. Springer, 2005.
10. Tok Wang Ling and W.B.T. Lee. DO2: A deductive object-oriented database system. In *Proceedings of the 9th International Conference on Database and Expert System Applications*, pages 50–59, 1998.
11. Tok Wang Ling and P.K. Teo. Inheritance conflicts in object-oriented systems. In *DEXA*, pages 189–200, 1993.
12. Mengchi Liu. The ROL deductive object base language. In *Proceedings of Database and Expert Systems Application*, pages 189–200, 1993.
13. Mengchi Liu. A logical foundation for XML. In *CAiSE*, pages 568–583, 2002.
14. Mengchi Liu and Tok Wang Ling. Towards declarative XML querying. In *Proceedings of WISE*, pages 127–138, Singapore, 2002.
15. G.Z. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *CoopIS/DOA/ODBASE*, pages 671–688, 2003.
16. Wei Zhang. XDO2: An XML deductive object-oriented query language. Master's thesis, School of Computing, National University of Singapore, 2004.

A XDO2 Language Syntax

Let $\mathbb{U}$ be a set of URLs, $\mathbb{C}$ be a set of constants, and $\mathbb{V}$ be a set of variables. The set of constants $\mathbb{C}$ contain strings enclosed by " ", integers, real numbers, two boolean values and object identifiers enclosed by ' '. The set of variables $\mathbb{V}$ are partitioned into single-valued and multi-valued variables. Single-valued variables have format $S where S is a string literal. Multi-valued variables include set-valued variables with format {$S} and list-valued variables with format <$S> where S is a string literal.

Definition 1. *The* values *are defined as follows,*

1. *null is a* null *value.*

2. *if* $c \in \mathbb{C}$ *then* c *is a* constant *value.*
3. *a set of object ids is a* set *value.*
4. *a list of constant values is a* list *value.*

Definition 2. *The* terms *are recursively defined as follows,*

1. *Let* t *be an XML attribute name. Then* $@t$ *is an* attribute *term.*
2. *Let* t *be an XML element tag. Then* t *is an* element *term.*
3. *Let* X *be an attribute name or a single-valued variable, and* Y *a constant value, a set value, a single-valued variable or a set-valued variable. Then* $@X : Y$ *is an* attribute_value *term, and* Y *denotes the value of the attribute* X*.*
4. *Let* X *be an element tag or a single-valued variable, and* Y *a constant value, a list value, a single-valued variable or a list-valued variable. Then* $X : Y$ *is an* element_value *term, and* Y *denotes the value of the element* X*.*
5. *Let* X *be a term. Then not(*X*) is a* negation *term.*
6. *Let* $X_1, \ldots, X_n$*,* $(n \geq 2)$ *be a set of terms. Then* $[X_1, \ldots, X_n]$ *is a* grouping *term.*
7. *Let* $X_1, \ldots, X_n$*,* $(n \geq 2)$ *be a set of terms where* $X_1, \ldots, X_{n-1}$ *are either element terms or element_value terms. Then* $X_1/\ldots/X_n$ *is a* path *term.*

Definition 3. *The* expressions *are defined as follows,*

1. *Let* $u \in \mathbb{U}$ *be a URL and* P *be a path term. Then* $(u)/P$ *is an* absolute path *expression.*
2. *Let* X *be a variable or an object id, and* P *be a term. Then* X/P *is a* relative path *expression. An* instantiable relative path *expression is a relative path expression* X/P *where either* X *is some object id, or the variable* X *has been defined in a positive term (i.e. not negation term).*
3. Arithmetic, logical *expressions are defined using variables, values, aggregate functions and operators in the usual way.* Instantiable arithmetic, logical *expressions are arithmetic, logical expressions such that all the variables inside have been defined in a positive term.*

Definition 4. *A* deductive rule *has the form* H *:-* $L_1, \ldots, L_n$*. where* H *is the head and* $L_1, \ldots, L_n$ *is the body of the rule.* H *is a positive instantiable relative path expression and* $L_1, \ldots, L_n$ *are either absolute path expressions or instantiable expressions.*

Definition 5. *A* query *has the form* $R \Leftarrow L_1, \ldots, L_n$*. where* R *is the result format expression and* $L_1, \ldots, L_n$ *are the query or conditional expressions.* R *is a positive absolute path expression and* $L_1, \ldots, L_n$ *are either absolute path expressions or instantiable expressions. If there is no result format expression specified, we use* $\Leftarrow L_1, \ldots, L_n$*.*

B XDO2 Language Semantics

For the language semantics, please refer to [16].