XML Declarative Description with Negative Constraints

Chutiporn Anutariya[1], Vilas Wuwongse[2] and Kiyoshi Akama[3]

[1] Department of Telematics, Norwegian University of Science and Technology
7491 Trondheim, Norway
Chutiporn.Anutariya@item.ntnu.no
[2] Computer Science & Information Management Program,
School of Advanced Technologies, Asian Institute of Technology
Pathumtani 12120, Thailand
vw@cs.ait.ac.th
[3] Center for Information and Multimedia Studies,
Hokkaido University, Sapporo 060, Japan
akama@cims.hokudai.ac.jp

Abstract

XML Declarative Description (*XDD*) theory employs XML's nested tree structure as its underlying data structure and *Declarative Description* theory as a framework to enhance its expressiveness. It enables direct representation of data items, encoded in XML applications, and extends their expressiveness by materializing succinct and uniform representation of implicit information, integrity constraints, conditional relationships and axioms. With the aim of enhancing XDD's expressive power, this paper presents its extension with a well-defined mechanism for formulation of *negative constraints* (or *negation*). In addition, the formulation is employed to treat arbitrary *XML first-order formulae* in terms of corresponding *XML first-order logical constraints* (*FLCs*)—special kind of constraints which comprise *XML expressions* and *logical symbols*: ¬, ∧, ∨, ⇒, ∀ and ∃. Despite its expressiveness, XDD retains precise and well-defined declarative semantics. It achieves sound, efficient and flexible computation by means of *Equivalent Transformation* (*ET*)—a new computational model, based on semantic-preserving transformations. Basic ET rules for manipulation and reasoning with negative constraints are developed and certain examples illustrated.

1. Introduction

XML—a description format for encoding and exchange of structured data and documents on the Web—lacks a computational mechanism and expressive power by not allowing uniform expression of domain knowledge, axioms, conditional relationships and constraints. Important aproaches to the development of XML-based information representation by enhancement of XML's expressiveness and provision of an efficient computational mechanism can be classified based on the three main features:

- *Document- and data-structure-based*: Approaches in this category are concerned with structural modeling of XML documents by means of various data-structure techniques such as *trees*, *graphs* (Abiteboul, Buneman and Suciu 2000) and *hedge automata* (Murata 1995; Murata 1997), which view a document as a tree, a graph or a hedge, respectively. However, they only provide mechanisms to define certain forms of structural relationships and constraints, while lack a facility to specify rules, conceptual relationships and axioms. In addition, queries about implicit information are not yet supported, since their sole querying mechanisms are based on structural and pattern matching.
- *Rule-based*: There exist several XML-based rule languages, developed for various purposes and applications, e.g., *BRML* (Grosof, Labrou and Chan 1999) and *XRML* (Lee and Sohn 2002). Most, if not all, of these languages are mere XML encoding of particular, existing logic programming and knowledge representation. Although they extend XML's capabilities by incorporation of rule-expression ability and a reasoning mechanism, their expressiveness is still limited by their inability to directly represent arbitrary XML elements. Extra tasks of schema and data conversions have to be performed to eliminate the mismatch in encoding between the rules and the documents to be processed. Furthermore, these rule languages lack theoretical or semantic foundations, unless they are translated back into their original language formalisms.
- *Ontology-based*: Major, current ontology-based markup languages, e.g., *DAML* (Hendler and McGuinness 2000), extend *RDF* (Lassila and Swick 1999) and *RDF Schema* (Brickley and Guha 2000) by providing further support for modeling ontologies with well-defined semantics and reasoning services. However, they still lack capabilities beyond those predefined primitives for definition of arbitrary ontological axioms. Moreover, their semantics involves mapping of their modeling primitives onto corresponding representations in a particular logical theory followed by corresponding determination of their semantics.

In summary, all these approaches require additional formalisms with definitions of semantics or specification of relationships and constraints.

XML Declarative Description (*XDD*) theory (Wuwongse et al. 2001; Wuwongse et al. 2003) is proposed with emphasis on the development of an XML-based knowledge representation, which provides in a single formalism a simple, yet expressive mechanism for succinct and uniform representation of explicit and implicit information, application axioms and constraints. It exploits XML's bare syntax and enhances XML's expressiveness by means of *Declarative Description* theory (Akama 1993; Akama, Shimitsu and Miyamoto 1998). XDD's basic modeling elements are ordinary XML elements, which can readily be used to represent *explicit* complex entities and their relationships in a real application domain. Moreover, it extends this capability of ordinary XML elements by additionally allowing representation of *implicit* complex entities as well as their classes, relationships, rules and constraints in terms of *XML expressions*—a generalization of XML elements with variables—and *constrained XML clauses*. An *XDD description* is formulated as a set of ordinary XML elements, XML expressions with variables and XML clauses. Its declarative semantics is formally defined as a set of ordinary XML elements which are directly described by or derivable from the description itself.

This paper enhances the expressive power of normal XDD descriptions by generalizing the concept of ordinary XML constraints into *referential XML constraints*. This allows an XDD description to refer to other XDD descriptions in order to represent/enforce (higher-order) complex relations and restrictions. The meaning of a given referential constraint is formally defined on the basis of the meaning of its referred descriptions.

Appropriate definition of *negative constraints*, formalized as a referential constraints, yields a well-defined mechanism for formulation of complex statements which can declaratively describe negative information or *negation*. Other types of referential constraints for capturing particular relations among XDD descriptions can also be devised. *Set-of constraint* (Akama et al. 2002), for instance, presents another significant constraint useful for defining set construction, set manipulation and aggregate functions.

Since XDD focuses on information/knowledge representation, in order to provide a concise and expressive language with precise and well-defined semantics, its underlying representation scheme is separated from its computational mechanism. It achieves efficient manipulation of and reasoning with XDD descriptions by employment of *Equivalent Transformation* (*ET*) (Akama, Shimitsu and Miyamoto 1998) computational paradigm.

The fundamental concepts of XDD theory recalled in Sect. 2 are formally extended with referential constraints in Sect. 3. Their declarative semantics and its employment to formalize XDD descriptions with negative constraints are also given. Sect. 4 presents their computational mechanism by means of ET paradigm and devises basic ET rules for negative constraints. Founded on the developed theory, Sect. 5 demonstrates its application to represent and reason about arbitrary first-order logical constraints. Sect. 6 concludes and presents further research direction.

2. XDD: An Informal Review

XDD extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressiveness and representation of implicit information into so-called *XML expressions*. Ordinary XML elements (variable-free XML expressions) are called *ground XML expressions*. Every component of an XML expression can contain variables, e.g., its expression or a sequence of sub-expressions (*E-variables*), tag names or attribute names (*N-variables*), strings or literal contents (*S-variables*), pairs of attributes and values (*P-variables*) and some partial structures (*I-variables*). Every variable is prefixed by '\$$T$:', where T denotes its type. For example, $S:value and $E:expression are *S*- and *E*-variables, which can be specialized into a string or a sequence of XML expressions, respectively. The data structure and specialization behavior of XML expressions are characterized by a mathematical abstraction $\langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{S}_X, \mu_X \rangle$, called *XML specialization system*, where

- $\mathcal{A}_X$ is the set of all XML expressions,
- $\mathcal{G}_X$ is the subset of $\mathcal{A}_X$ comprising all ground XML expressions,
- $\mathcal{S}_X$ is the set of *XML specializations*, each of which specifies a (possibly null) sequence of variables to be specialized and their specializing values/patterns,
- μ_X is the *specialization operator* which determines, for each XML specialization θ in $\mathcal{S}_X$, the change of an XML expression in $\mathcal{A}_X$ caused by θ.

Unless confusion may arise, the application of a specialization θ to an XML expression a by the operator μ_X, formally denoted by $\mu_X\theta(a)$, will be represented as $a\theta$.

Basically, objects and their simple relationships are explicitly represented as ground XML expressions, while a more complex and possibly implicit one is modeled by an XML clause C of the form: $H \leftarrow B_1, \ldots, B_n$, where $n \geq 0$, H is an XML expression, and B_i is an XML expression or an *XML constraint*. H is called the *head* and $\{B_1, \ldots, B_n\}$ the *body* of the clause. When its body is the empty set, C will be referred to as an *XML unit clause* and the symbol $\leftarrow$ is often omitted; hence, an XML element or document is mapped directly onto a *ground XML unit clause*.

An *XML constraint* has the form $\phi(a_1, \ldots, a_n)$, where $n > 0$, ϕ is a *constraint predicate* and a_i is an XML expression. The satisfaction (truth or falsity) of a *ground constraint* is predetermined. When the context is clear, some constraints will be represented using infix notation. For instance, [$S:salary > 10000] and [$S:bonus := $S:salary * 2] represent the constraints GT($S:salary, 10000) and Mul($S:salary, 2, $S:bonus), respectively.

An *XDD description* is a set of XML clauses. Intuitively, given an XDD description P, its meaning denoted by $\mathcal{M}(P)$ is the set of all XML elements which are directly described by or are derivable from the unit and non-unit clauses in P. These elements are *surrogates* of real, tangible or intangible objects or their relationships in the domain of interest, modeled by the description P.

```
E1: <hr:Staff  rdf:about="john">
        <hr:name>John Smith</hr:name>
        <hr:education>Doctor</hr:education>
        <hr:department  rdf:resource="sales"/>
        <hr:position>Sales Manager</hr:position>
        <hr:salary>12000</hr:salary>
    </hr:Staff>
E2: <hr:Staff  rdf:about="jack">
        <hr:name>Jack White</hr:name>
        <hr:education>Master</hr:education>
        <hr:department  rdf:resource="computer"/>
        <hr:position>Engineer</hr:position>
        <hr:salary>7000</hr:salary>
    </hr:Staff>
E3: <hr:Department  rdf:about="sales">
        <hr:name>Sales Department</hr:name>
        <hr:telephone>750 2255<hr:telephone>
    </hr:Department>
E4: <hr:Department  rdf:about="computer">
        <hr:name>Computer Department</hr:name>
        <hr:telephone>750 2277<hr:telephone>
    </hr:Department>
C1: <hr:SeniorStaff  rdf:about=$S:X>
        <hr:salary>$S:salary</hr:salary>
        <hr:bonus>$S:bonus</hr:bonus>
        $E:Xproperties
    </hr:SeniorStaff>
        ←    <hr:Staff  rdf:about=$S:X>
                <hr:salary>$S:salary</hr:salary>
                $E:Xproperties
             </hr:Staff>,
             [$S:salary > 10000],
             [$S:bonus := $S:salary * 2].
C2: <hr:DepartmentWithSeniorStaff  rdf:about=$S:dept>
        $E:deptProperties
    </hr:DepartmentWithSeniorStaff>
        ←    <hr:Department  rdf:about=$S:dept>
                $E:deptProperties
             </hr:Department>,
             <hr:SeniorStaff  rdf:about=$S:staff>
                <hr:department  rdf:resource=$S:dept/>
                $E:staffProperties
              </hr:SeniorStaff>.
```

a. An XDD description *XDB* = {*E*$_1$, *E*$_2$, *E*$_3$, *E*$_4$, *C*$_1$, *C*$_2$}

```
E5: <hr:SeniorStaff  rdf:about="john">
        <hr:name>John Smith</hr:name>
        <hr:education>Doctor</hr:education>
        <hr:department  rdf:resource="sales"/>
        <hr:position>Sales Manager</hr:position>
        <hr:salary>12000</hr:salary>
        <hr:bonus>24000</hr:bonus>
    </hr:SeniorStaff>
E6: <hr:DepartmentWithSeniorStaff  rdf:about="sales">
        <hr:name>Sales Department</hr:name>
        <hr:telephone>750 2255<hr:telephone>
    </hr:DepartmentWithSeniorStaff>
```

b. XML elements derived from XDB and included in $\mathcal{M}(XDB)$

Figure 1. An XDD description example and its meaning

Example 1 The XDD description XDB of Fig. 1-a presents an example of modeling an RDF/XML database of a simple human resource management application. It comprises four unit clauses $E_1 - E_4$, representing XML elements in the database, and two non-unit clauses: C_1 and C_2, formalizing application axioms for the concepts SeniorStaff and DepartmentWithSeniorStaff. C_1 defines that if X is a Staff whose salary is more than 10000, then X is regarded as a SeniorStaff and will receive a double-salary bonus. C_2 defines that a department $S:dept is a DepartmentWithSeniorStaff if there is a SeniorStaff working for that department. The meaning of XDB is the set of XML elements which are directly described by the unit clauses, i.e., the elements $E_1 - E_4$, together with those which are deducible from the database, i.e., the elements E_5 and E_6 of Fig. 1-b. ❐

3. Referential XDD Descriptions

Here, XDD descriptions will be extended with the ability to represent *referential XML constraints*. In essence, referential constraints generalize the concept of ordinary XML constraints by inclusion of a facility for referring to other XDD descriptions, allowing representation of (higher-order) relations or restrictions. The meaning of a given referential constraint is determined based on the meanings of its referred descriptions. Thus, by inclusion of referential constraints in the clause's body, an XML clause can additionally represent/enforce complex relations/ restrictions, the truth or falsity of which is evaluated according to the meaning of particular XDD descriptions.

The definitions of XML constraints, XML clauses and XDD descriptions are redefined:

Definition 1 XML constraints, XML clauses and **XDD descriptions** are defined inductively by:

1. An *XML constraint* is an $(n+m+1)$-tuple $\langle \phi, a_1, \ldots, a_n, Q_1, \ldots, Q_m \rangle$, where
   - $n, m \geq 0$,
   - ϕ is a *mapping* from $G_1 \times \ldots \times G_n \times G_{n+1} \times \ldots \times G_{n+m}$ to $\{true, false\}$, where G_i is $\mathcal{G}_X$ for each $1 \leq i \leq n$, and $2^{\mathcal{G}_X}$ for each $n+1 \leq i \leq n+m$,
   - a_i is an XML expression in $\mathcal{G}_X$,
   - Q_i is an XDD description.
2. An *XML clause* is a *formula* $(H \leftarrow B_1, \ldots, B_n)$, where $n \geq 0$, H is an XML expression, and B_i is either an XML expression or an XML constraint.
3. An *XDD description* is a *set* of XML clauses. ❐

An XML constraint $\langle \phi, a_1, \ldots, a_n, Q_1, \ldots, Q_m \rangle$ is called a *simple XML constraint* iff it comprises solely XML expressions (i.e., $m = 0$), and called a *referential XML constraint* iff it refers to some XDD descriptions (i.e., $m \geq 1$). Moreover, it is a *ground XML constraint* iff it comprises only ground XML expressions and XDD descriptions. A clause C is a *ground clause* iff it comprises only ground XML expressions and ground XML constraints. The head of C is denoted by $head(C)$ and the set of all XML expressions and XML constraints in the body of C by $object(C)$ and $con(C)$, respectively. Let $body(C)=object(C) \cup con(C)$. Given $\theta \in \mathcal{S}_X$, application of θ to a clause $(H \leftarrow B_1, \ldots, B_n)$ yields the clause $(H\theta \leftarrow B_1\theta, \ldots, B_n\theta)$, and to a constraint $\langle \phi, a_1,\ldots, a_n, Q_1,\ldots, Q_m\rangle$ the constraint $\langle \phi, a_1\theta, \ldots, a_n\theta, Q_1, \ldots, Q_m\rangle$.

Employing the formalized concepts, a *negative constraint*—one of the most important referential XML constraints—will be given for representation of negative information or *negation*.

Definition 2 A **negative constraint** has the form

$$\langle f_{not}, a, Q\rangle,$$

where – f_{not} is a mapping from $\mathcal{G}_X \times 2^{\mathcal{G}_X}$ to $\{true, false\}$ such that
- $f_{not}(g, G) = true$ if $g \notin G$,
- $f_{not}(g, G) = false$ if $g \in G$,

– a is an XML expression,
– Q an XDD description. ❒

A constraint $\langle f_{not}, a, Q\rangle$ restricts that the expression a must not be true with respect to the description Q, i.e., a must not be included in the meaning of Q. In the sequel, such a constraint will be simply represented as $[a \notin \mathcal{M}(Q)]$. This formulation of negation is natural because it permits specification of negative information and its respective evaluation context.

Example 2 By referring to the description XDB of Fig. 1, the following clause C_{neg} derives those departments with no SeniorStaff.

```
Cneg: <DepartmentWithoutSeniorStaff  rdf:about=$S:dept>
        $E:deptProperties
      </DepartmentWithoutSeniorStaff>
    ←    <hr:Department  rdf:about=$S:dept>
           $E:deptProperties
         </hr:Department>,
         [<hr:DepartmentWithSeniorStaff
              rdf:about=$S:dept>
           $E:deptProperties
          </hr:DepartmentWithSeniorStaff> ∉ M(XDB) ].
```

It simply specifies that a department $S:dept is a Department-WithoutSeniorStaff if it is not a DepartmentWithSeniorStaff. The first XML expression in C_{neg}'s body finds a Department-element from the database and the negative constraint specifies that such a department is not a DepartmentWith-SeniorStaff with respect to the description XDB. C_{neg}'s head then derives that such a department is a DepartmentWithout-SeniorStaff. ❒

Definition 3 The **declarative semantics** of an XDD description P, denoted by $\mathcal{M}(P)$, is defined inductively by:

1. Given the meanings $\mathcal{M}(Q_1), \ldots, \mathcal{M}(Q_m)$ of XDD descriptions $Q_1, \ldots, Q_m$, a ground constraint $\langle \phi, g_1, \ldots, g_n, Q_1, \ldots, Q_m\rangle$ is a true XML constraint iff $\phi(g_1, \ldots, g_n, \mathcal{M}(Q_1), \ldots, \mathcal{M}(Q_m)) = true$. Define the set $Tcon$ as the set of all true ground XML constraints, i.e.:

$$Tcon = \{\langle \phi, g_1, \ldots, g_n, Q_1, \ldots, Q_m\rangle \mid g_i \in \mathcal{G}_X, Q_i \text{ is an XDD description}, \phi(g_1,\ldots,g_n, \mathcal{M}(Q_1),\ldots,\mathcal{M}(Q_m)) = true\}.$$

2. The meaning $\mathcal{M}(P)$ of the description P is the set of ground XML expressions defined by:

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} [T_P]^n(\varnothing),$$

where – $T_P^1(\varnothing) = T_P(\varnothing)$,
– $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$ for each $n > 1$,
– the mapping T_P: $2^{\mathcal{G}_X} \rightarrow 2^{\mathcal{G}_X}$ is defined by:

For each $G \subset \mathcal{G}_X$, a ground XML expression g is contained in $T_P(G)$ iff there exist an XML clause $C \in P$ and an XML specialization $\theta \in \mathcal{S}_X$ such that $C\theta$ is a ground clause with the head g, all the XML expressions and constraints in the body of which belong to G and $Tcon$, respectively, i.e.:

$$T_P(G) = \{ head(C\theta) \mid C \in P, \theta \in \mathcal{S}_X, C\theta \text{ is a ground clause}, object(C) \subset G, con(C) \subset Tcon\}. \text{ ❒}$$

From its definition, one can yield that the meaning of a description is defined based on the meanings of its referred descriptions, and thus XDD descriptions with referential constraints must be *stratified*. Speaking intuitively, the meaning of P, i.e., $\mathcal{M}(P)$, is a set of all ground XML expressions which are directly described by and are derivable from the unit and the non-unit XML clauses in P.

4. Computation Mechanism

4.1. Equivalent Transformation

Computation of XDD descriptions is carried out by employment of *Equivalent Transformation* (*ET*) (Akama, Shimitsu and Miyamoto 1998)—a new, flexible and efficient computational model which solves a given problem, described in an appropriate language, by simplifying it through repetitive application of (semantically-)equivalent transformation rules. Let P_1 be an XDD description which models a particular application or problem. The meaning of P_1, i.e., $\mathcal{M}(P_1)$, yields the set of XML elements which represent concepts, instances and interrelationships of these objects in the application domain, and hence yields the solutions to the formulated problem. This new computation paradigm applies *ET rules* (procedural rewriting rules) in order to successively transform P_1 into P_2, P_3, etc., while maintaining the conditions $\mathcal{M}(P_1) = \mathcal{M}(P_2) = \mathcal{M}(P_3) = \ldots$, until the desirable description P_n, which is simpler but equivalent description from which the answers to the given problem specification can be drawn easily and directly, is obtained. More precisely, P_1 is successively transformed until it becomes the description P_n, where P_n holds only XML unit clauses and $\mathcal{M}(P_1) = \mathcal{M}(P_n)$. Hence, the XML elements directly described by these unit clauses in P_n

readily represent the answers to the problem described by P_1. Moreover, since only ET rules are applied in each transformation step, the meanings of the descriptions are maintained and the correctness of the computation is always guaranteed. The unfolding transformation—the fundamental computational mechanism in logic programming—presents an example of ET rules. There exist many other ET rules which reflect or exploit specific application domain knowledge and data structure, and thus could lead to more efficient computation.

Recall that an XDD description can refer to other descriptions by means of referential XML constraints. These descriptions, thus, form a directed tree structure, where a description is a parent node of those descriptions it refers to, and it is a child node of those descriptions referring to it. Computation of each description is carried out in its own *world* in parallel, where a child node (description) is computed in a *child world* and a parent node is computed in a *parent world*. Information can also be propagated to/from the parent and child worlds.

4.2 XML Equivalent Transformation

Founded on the XDD theory and the ET paradigm, *XML Equivalent Transformation* (*XET*) engine (Anutariya et al. 2002; Anutariya, Wuwongse and Wattanapailin 2002) has been developed for direct and succinct manipulation of and reasoning with XML expressions without a necessity for data conversion. An XET program comprises a set of *XET rules* and a set of XML elements/documents, regarded as the program's data or *facts*. These rules and facts are prepared (semi-automatically) according to the specification provided by the given XDD description. In general, given an XDD description modeling a problem in a particular domain, a set of XET rules and facts for implementing such a problem can be easily obtained. Thus, XDD descriptions are viewed as XET program specifications. Additional XET rules for improvement of computation efficiency can also be devised based on certain specific characteristics and properties of application data and rules. The general form of an XET rule is

$$\begin{array}{lll} Head,\ \{Condition\} & \rightarrow & \{Execution_1\},\ Body_1; \\ & \rightarrow & \{Execution_2\},\ Body_2; \\ & & \quad\ldots \\ & \rightarrow & \{Execution_n\},\ Body_n; \end{array}$$

It specifies backward-chaining-like computation and reads: if the pattern of XML expressions specified by the $Head$ of a rule matches with the target XML expression and the $Condition$ is satisfied, then the n bodies fire simultaneously, i.e., the built-in or user-defined operations specified in each $Execution_i$ are performed, and the list of XML expressions specified in the $Body_i$ replace the target expression. More precisely, given an XML clause C: $H \leftarrow B, B_1, \ldots, B_m$, such an XET rule is applicable to the target expression B of the given clause C iff there exists a specialization θ such that $Head\theta = B$ (i.e., the target expression B must be more specific than the pattern specified by $Head$) and $Condition\theta$ is true (by a given evaluator). When applied, the rule transforms the clause C into (at most) n clauses: $C_1, \ldots, C_n$. Each clause C_i is obtained from C by executing $Execution_i\theta$ and replacing the target expression B of C by the XML expression(s) specified by $Body_i\theta$.

Example 3 Fig. 2 presents an XET program XDB.xet for reasoning with the XML database XDB of Fig. 1. To demonstrate the XET execution mechanism, consider a query which simply returns names of all SeniorStaff working for the sales department, formulated as the clause:

```
<answer>$S:name</answer>
    ←   <hr:SeniorStaff rdf:about=$S:X>
            <hr:name>$S:name</hr:name>
            <hr:department rdf:resource="sales"/>
            $E:properties
        </hr:SeniorStaff>.
```

Execution of such a query against the database XDB is carried out by transforming the query clause using the implemented rule and facts in XDB.xet. Firstly, the SeniorStaff rule (Lines 27–52) is applied because the hr:SeniorStaff-expression in the query clause's body can match with the head (Lines 28–32) of that rule. That is, the built-in operation xet:Unify (Lines 34–43) is executed and the hr:SeniorStaff-expression in the query clause's body is replaced by the expressions specified in the rule's body, i.e.: hr:Staff, xet:GT and xet:Mul (Lines 44–50). Hence, the clause is transformed into:

```
<answer>$S:name</answer>
    ←   <hr:Staff rdf:about=$S:X>
            <hr:name>$S:name</hr:name>
            <hr:department rdf:resource="sales"/>
            <hr:salary>$S:salary</hr:salary>
            $E:StaffProperties
        </hr:Staff>,
        <xet:GT number1=$S:salary number2="10000"/>,
        <xet:Mul number=$S:salary multiplier="2"
                result=$S:bonus/>.
```

Since the hr:Staff-expression in the clause's body can be matched successfully with the first fact (Lines 4–10) in the program, such an expression is removed and the clause is then transformed into:

```
<answer>John Smith</answer>
    ←   <xet:GT number1="12000" number2="10000"/>,
        <xet:Mul number="12000" multiplier="2"
                result=$S:bonus/>.
```

After the execution of the built-in operations xet:GT and xet:Mul, the answer to the query is obtained:

```
<answer>John Smith</answer>  ←   .
```

That is, John Smith is the only SeniorStaff working for the sales department. ❐

4.3. Rules for Negative Constraints

This section defines three basic ET rules for computation of XDD descriptions with negative constraints.

```
<xet:Program  xmlns:xet="http://kr.cs.ait.ac.th/XET"
  xmlns:hr="http://kr.cs.ait.ac.th/hr-schema.daml">
   <xet:Fact>
      <hr:Staff  rdf:about="john">
         <hr:name>John Smith</hr:name>
         <hr:education>Doctor</hr:education>
         <hr:department  rdf:resource="sales"/>
         <hr:position>Sales Manager</hr:position>
         <hr:salary>12000</hr:salary>
      </hr:Staff>
      <hr:Staff  rdf:about="jack">
         <hr:name>Jack White</hr:name>
         <hr:education>Master</hr:education>
         <hr:department  rdf:resource="computer"/>
         <hr:position>Engineer</hr:position>
         <hr:salary>7000</hr:salary>
      </hr:Staff>
      <hr:Department  rdf:about="sales">
         <hr:name>Sales Department</hr:name>
         <hr:telephone>750 2255<hr:telephone>
      </hr:Department>
      <hr:Department  rdf:about="computer">
         <hr:name>Computer Department</hr:name>
         <hr:telephone>750 2277<hr:telephone>
      </hr:Department>
   </xet:Fact>
   <xet:Rule name="SeniorStaff">
      <xet:Head>
         <hr:SeniorStaff  rdf:about="Svar_X">
            Evar_SeniorStaffProperties
         </hr:SeniorStaff>
      </xet:Head>
      <xet:Body>
         <xet:Unify>
            <xet:Expression>
               Evar_SeniorStaffProperties
            </xet:Expression>
            <xet:Expression>
               <hr:salary>Svar_salary</hr:salary>
               <hr:bonus>Svar_bonus</hr:bonus>
               Evar_StaffProperties
            </xet:Expression>
         </xet:Unify>
         <hr:Staff  rdf:about="Svar_X">
            <hr:salary>Svar_salary</hr:salary>
            Evar_StaffProperties
         </hr:Staff>
         <xet:GT  number1="Svar_salary"  number2="10000"/>
         <xet:Mul number="Svar_salary" multiplier="2"
                 result="Svar_bonus"/>
      </xet:Body>
   </xet:Rule>
   <xet:Rule name="DepartmentWithSeniorStaff">
      <xet:Head>
         <hr:DepartmentWithSeniorStaff rdf:about="Svar_dept">
            Evar_deptProperties
         </hr:DepartmentWithoutSeniorStaff>
      </xet:Head>
      <xet:Body>
         <hr:Department  rdf:about="Svar_dept">
            Evar_deptProperties
         </hr:Department>
         <hr:SeniorStaff  rdf:about="Svar_staff">
            <hr:department  rdf:resource="Svar_dept"/>
            Evar_staffProperties
         </hr:SeniorStaff>.
      </xet:Body>
   </xet:Rule>
</xet:Program>
```

Figure 2. XET program XDB.xet

Throughout this section, let P, Q, Q' and Q'' be XDD descriptions, C and C' be XML clauses, and a and b be XML expressions. Firstly, a transformation, which defines the starting point for processing negative constraints, is presented. It merges the expression a of a given negative constraint $[a \notin \mathcal{M}(Q)]$ and its referred description Q into a new description $Q' = Q \cup \{$ <xet:NotRule>a</xet:NotRule> $\leftarrow a\ \}$, and then transforms the given constraint into [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')$]. Thus, closely interrelated computation with the expression a and the referred description Q becomes possible by equivalent transformation of the description Q'.

ET Rule 1 (Merging of Object) Let

– $Q' = Q \cup \{$<xet:NotRule>a</xet:NotRule> $\leftarrow a\}$,
– C: $(H \leftarrow B_1, \ldots, B_i, [a \notin \mathcal{M}(Q)], B_{i+1}, \ldots, B_n)$,
– C': $(H \leftarrow B_1, \ldots, B_i,$ [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')], B_{i+1}, \ldots, B_n)$.

Then, $\mathcal{M}(P \cup \{C\}) = \mathcal{M}(P \cup \{C'\})$, and $P \cup \{C\}$ can be transformed equivalently into $P \cup \{C'\}$. ❒

Next, let Q' allow equivalent transformation into $Q'' \cup \{$<xet:NotRule>b</xet:NotRule> $\leftarrow\ \}$. Hence, the negative constraint [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')$] can be replaced by $[a \notin rep(b)]$ and [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q'')$], where $[a \notin rep(b)]$ represents a constraint which will be true iff a is an XML element and a is not a ground instance of b, i.e., there must not exist $\theta \in \mathcal{S}_X$ such that $a = b\theta$. This process is materialized by the following transformation:

ET Rule 2 (Lifting Transformation) Let

– $Q' = Q'' \cup \{$<xet:NotRule>b</xet:NotRule> $\leftarrow\ \}$,
– C: $(H \leftarrow B_1, \ldots, B_i,$ [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')], B_{i+1}, \ldots, B_n)$,
– C': $(H \leftarrow B_1, \ldots, B_i, [a \notin rep(b)],$ [<xet:NotRule>a</xet:NotRule>$\notin \mathcal{M}(Q'')],$ $B_{i+1}, \ldots, B_n)$,

where $[a \notin rep(b)]$ is a true constraint iff a is an XML element which is not a ground instance of b, i.e., there is no $\theta \in \mathcal{S}_X$ such that $a = b\theta$.

Then, $\mathcal{M}(P \cup \{C\}) = \mathcal{M}(P \cup \{C'\})$, and $P \cup \{C\}$ can be transformed equivalently into $P \cup \{C'\}$. ❒

If the referred description Q' of a constraint [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')$] cannot be transformed equivalently into a description $Q'' \cup \{$<xet:NotRule>b</xet:NotRule> $\leftarrow\ \}$, such a constraint can be eliminated from a clause. This is implemented by:

ET Rule 3 (Elimination of Negative Constraint) Let

– Q' be a description which cannot be transformed equivalently into a description $Q'' \cup \{$<xet:NotRule>b</xet:NotRule> $\leftarrow\ \}$,
– C: $(H \leftarrow B_1, \ldots, B_i,$ [<xet:NotRule>a</xet:NotRule> $\notin \mathcal{M}(Q')], B_{i+1}, \ldots, B_n)$,

– C': $(H \leftarrow B_1, \ldots, B_i, B_{i+1}, \ldots, B_n)$.

Then, $\mathcal{M}(P \cup \{C\}) = \mathcal{M}(P \cup \{C'\})$, and $P \cup \{C\}$ can be transformed equivalently into $P \cup \{C'\}$. ❐

Example 4 Based on the presented three rules for negative constraints and the implemented rules and facts of XDB.xet (Fig. 2), the computation of the query C_{neg} (Example 2), which finds departments with no SeniorStaff, is shortly demonstrated.

Step 1: The original problem is formalized as:

```
P = {C_neg:  <DepartmentWithoutSeniorStaff  rdf:about=$S:dept>
                 $E:deptProperties
             </DepartmentWithoutSeniorStaff>
             ⟵   <hr:Department  rdf:about=$S:dept>
                          $E:deptProperties
                 </hr:Department>,
                 [<hr:DepartmentWithSeniorStaff
                         rdf:about=$S:dept>
                         $E:deptProperties
                 </hr:DepartmentWithSeniorStaff> ∉ M(XDB)]}
```

Step 2: Transform the negative constraint in C_{neg} by ET Rule 1:

```
P = {C_neg: <DepartmentWithoutSeniorStaff  rdf:about=$S:dept>
                 $E:deptProperties
            </DepartmentWithoutSeniorStaff>
             ⟵   <hr:Department  rdf:about=$S:dept>
                         $E:deptProperties
                 </hr:Department>,
                 [<xet:NotRule>
                     <hr:DepartmentWithSeniorStaff
                         rdf:about=$S:dept>
                         $E:deptProperties
                     </hr:DepartmentWithSeniorStaff>
                  </xet:NotRule>  ∉ M(XDB′)] }
XDB′ = XDB ∪ {<xet:NotRule>
                     <hr:DepartmentWithSeniorStaff
                         rdf:about=$S:dept>
                         $E:deptProperties
                     </hr:DepartmentWithSeniorStaff>
                  </xet:NotRule>
                     ⟵   <hr:DepartmentWithSeniorStaff
                             rdf:about=$S:dept>
                             $E:deptProperties
                         </hr:DepartmentWithSeniorStaff> }
```

Step 3: By applying the rules DepartmentWithSeniorStaff (Lines 27-51) and SeniorStaff (Lines 52-67) as well as unfolding with the facts (Lines 3-26) in XDB.xet, XDB' can be equivalently transformed into:

```
XDB′ = XDB ∪ {<xet:NotRule>
                     <hr:DepartmentWithSeniorStaff
                         rdf:about="sales">
                       <hr:name>Sales Department</hr:name>
                       <hr:telephone>750 2255<hr:telephone>
                     </hr:DepartmentWithSeniorStaff>
                  </xet:NotRule>  ⟵}
```

Step 4: Because XDB' comprises a unit clause with the xet:NotRule-expression in its head, ET Rule 2 is applied:

```
P = {C_neg:<DepartmentWithoutSeniorStaff   rdf:about=$S:dept>
                 $E:deptProperties
            </DepartmentWithoutSeniorStaff>
              ⟵   <hr:Department  rdf:about=$S:dept>
                         $E:deptProperties
                   </hr:Department>,
                  [<hr:Department  rdf:about=$S:dept>
                         $E:deptProperties
                   </hr:Department> ∉
                     rep(<hr:DepartmentWithSeniorStaff
                             rdf:about="sales">
                                <hr:name>Sales Department</>
                                <hr:telephone>750 2255</>
                             </hr:DepartmentWithSeniorStaff>)]
                    [<xet:NotRule>
                       <hr:DepartmentWithSeniorStaff
                           rdf:about=$S:dept>
                           $E:deptProperties
                       </hr:DepartmentWithSeniorStaff>
                     </xet:NotRule>  ∉ M(XDB)] }
```

Step 5: Because XDB' cannot be further transformed into a description comprising a unit clause with xet:NotRule-expression in the head, ET Rule 3 is applied:

```
P = {C_neg: <DepartmentWithoutSeniorStaff  rdf:about=$S:dept>
                 $E:deptProperties
             </DepartmentWithoutSeniorStaff>
              ⟵   <hr:Department  rdf:about=$S:dept>
                         $E:deptProperties
                   </hr:Department>,
                  [<hr:Department  rdf:about=$S:dept>
                         $E:deptProperties
                   </hr:Department>  ∉
                   rep(<hr:DepartmentWithSeniorStaff
                             rdf:about="sales">
                                <hr:name>Sales Department</>
                                <hr:telephone>750 2255</>
                           </hr:DepartmentWithSeniorStaff>)]}
```

Step 6: By unfolding the hr:Department-expression in C_{neg}'s body with the third and fourth facts of XDB.xet and by evaluating the constraint $[a \notin rep(b)]$ in the clause's body, the description P can be equivalently transformed into:

```
P = {C_neg:<DepartmentWithoutSeniorStaff   rdf:about="computer">
              <hr:name>Computer Department</hr:name>
              <hr:telephone>750 2277<hr:telephone>
           </DepartmentWithoutSeniorStaff> ⟵        }
```

Since only semantic-preserving transformations are used in every step, the meaning of the transformed description P is preserved and the obtained answer:

```
<DepartmentWithoutSeniorStaff   rdf:about="computer">
   <hr:name>Computer Department</hr:name>
   <hr:telephone>750 2277<hr:telephone>
</DepartmentWithoutSeniorStaff>
```

is guaranteed to be correct. ❐

5. Extended XDD Descriptions with FLCs

This section outlines an extension of XDD descriptions with the mechanism to express arbitrary first-order formulae. Such an expression is represented as a *first-order* *logical constraint* (*FLC*)—a special kind of constraints which is composed of XML expressions and logical symbols: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$ and $\exists$. An FLC is defined as:

1. An XML expression is an FLC.
2. An XML constraint is an FLC.
3. If F and G are FLCs, then so are $\neg F$, $F \wedge G$, $F \vee G$ and $F \Rightarrow G$.
4. If F is an FLC and x is a variable, then ($\forall x\ F$) and ($\exists x\ F$) are FLCs.

An XDD description with FLCs is then formalized as a set of extended XML clauses with FLCs. Each clause has the form $H \leftarrow F$, where the head H is an XML expression and the body F an FLC. Employing those transformation techniques developed in the first-order logic theory (Lloyd 1987), extended XDD descriptions with FLCs can be straightforwardly transformed into equivalent XDD descriptions (with negative constraints) in a finite number of steps, and the meaning of a given description with FLCs is directly defined by means of its equivalent description.

6. Conclusions

The developed XDD theory with negative constraints yields a unified XML-based knowledge representation with sufficient expressive power to represent all XML applications, enhances their expressiveness and lets their intended meanings be determined directly. It provides more direct insight into XML data representation and manipulation, and facilitates means for uniform expression of explicit and implicit information, ontologies, constraints, negation and axioms by direct employment of XML elements as its basic language component. It is employed to model various applications such as constrained XML databases (Akama et al. 2002; Wuwongse et al. 2003), the Semantic Web (Anutariya et al. 2002; Suwanapong, Anutariya and Wuwongse 2002; Wuwongse et al. 2001) and UML diagrams (Nantajeewarawat et al.). Employment of the ET framework leads to computation with XDD by successive, equivalent transformation of a given description—a problem's specification—into the desired one—a problem's solution. Hence, the efficiency of the computation relies solely on the employed ET rules.

XET engine (Anutariya et al. 2002; Anutariya, Wuwongse and Wattanapailin 2002), which integrates XML syntax, XDD representation style and ET mechanism, is also available for direct and succinct manipulation of and reasoning with XDD descriptions. Its employment to implement various XML-, RDF- and DAML-based prototype systems—such as an XML database system (Akama et al. 2002; Wuwongse et al. 2003), an intelligent, ontology-enabled Semantic Web service system (Suwanapong, Anutariya and Wuwongse 2002) and e-business applications (Anutariya et al. 2002; Anutariya, Wuwongse and Wattanapailin 2002)—helps demonstrate the viability and potential in real applications. Further work includes development of more efficient ET rules specific for XML data structure as well as incorporation of basic built-in ET rules into the XET engine for dealing with FLC constraints.

References

Abiteboul, S, Buneman, P., Suciu, D. 2000. *Data on the Web: From Relations to Semistructured Data and XML*: Morgan Kaufmann Publishers.

Akama, K. 1993. Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advances in Software Science and Technology* 5:45-63.

Akama, K., Shimitsu, T., Miyamoto, E. 1998. Solving Problems by Equivalent Transformation of Declarative Programs. *Journal of the Japanese Society of Artificial Intelligence* 13(6):944-952.

Akama, K., Anutariya, C., Wuwongse, V. and Nantajeewarawat, E. 2002. Query Formulation and Evaluation for XML Databases. In *Proc. First IFIP Workshop on Internet Technologies, Applications, and Societal Impact*, 273-288. Wroclaw, Poland: Kluwer Academic Publisher.

Anutariya, C., Wuwongse, V., Akama, K. and Wattanapailin, V. 2002. Semantic Web Modeling and Programming with XDD. *The Emerging Semantic Web, Series: Frontiers in Artificial Intelligence and Applications* 75: IOS Press.

Anutariya, C., Wuwongse, V. and Wattanapailin, V. 2002. An Equivalent-Transformation-Based XML Rule Language. In *Proc. International Workshop on Rule Markup Languages for Business Rules in the Semantic Web*, Italy.

Brickley, D. and Guha, R. V. 2000. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation March 2000.

Grosof, B.N., Labrou, Y. and Chan, H.Y. 1999. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proc. First ACM Conference on Electronic Commerce (EC99)*, 68–77.

Hendler, J. and McGuinness, D.L. 2000. The DARPA Agent Markup Language. *IEEE Intelligent Systems* 15(6):72-73.

Lassila, O. and Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation February 1999. [http://www.w3.org/TR/REC-rdf-syntax]

Lee, J.K. and Sohn, M.M.. 2002. Extensible Rule Markup Language – Toward the Intelligent Web Platform. *Communications of the ACM*. Forthcoming.

Lloyd, J. W. 1987. *Foundations of Logic Programming*, 2nd Edition: Springer Verlag.

Murata, M. 1995. Hedge Automata: A Formal Model for XML Schemata, Technical Report, Fiji Xerox Information Systems.

Murata, M. 1997. Transformation of Documents and Schemas by Patterns and Contextual Conditions. In *Proc. Principle of Document Processing*, LNCS 1239, 153-159: Springer Verlag.

Nantajeewarawat, E., Wuwongse, V., Anutariya, C., Akama, K. and Thiemjarus, A. Towards Reasoning with UML Diagrams Based-on XML Declarative Description Theory. *Journal of Intelligent Systems*. Forthcoming.

Suwanapong, S., Anutariya , C. and Wuwongse, V. 2002. An Intelligent Web Service System. In *Proc. of IFIP WG8.1 Working Conf. Information Systems in the Internet Context* (*EISIC'02*), Kanazawa, Japan: Kluwer Academic Publisher.

Wuwongse, W., Akama, K., Anutariya, C. and Nantajeewarawat, E. 2003. A Data Model for XML Databases. *Journal of Intelligent Information Systems* 20(1):63-80.

Wuwongse, W., Anutariya, C., Akama, K. and Nantajeewarawat, E.: XML Declarative Description (XDD): A Language for the Semantic Web. *IEEE Intelligent Systems* 16(3):54-65.