XPathLog: A Declarative, Native XML Data Manipulation Language

Wolfgang May
Institut für Informatik, Universität Freiburg, Germany
may@informatik.uni-freiburg.de

Abstract
XPathLog is a logic-based language for manipulating and integrating XML data. It extends the XPath query language with Prolog-style variables. Due to the close relationship with XPath, the semantics of rules is easy to grasp. In contrast to other approaches, the XPath syntax and semantics is also used for a declarative specification how the database should be updated: when used in rule heads, XPath filters are interpreted as specifications of elements and properties which should be added to the database. The formal semantics is defined wrt. a graph Herbrand structure which covers the XML tree data model. XPathLog has been implemented in LoPiX.

1 Introduction
XML has been designed and accepted as the framework for semi-structured data where it plays the same role as the relational model for classical databases. Specialized languages are available for XML querying, e.g., XQL [13], XML-QL [5], Quilt/XQuery [3, 19] and for transformations of XML sources, e.g., XSL(T) [20] (also XML-QL and XQuery can be used for generating new XML documents from given ones since their output format is XML), but yet none of them can be seen as an XML data manipulation language. For writing applications for creating and manipulating XML data, the dominating language is Java, regarding the DOM model as a data structure where applications are built on.

We propose a declarative, Prolog-style language for manipulation and integration of XML documents. The syntax and querying semantics is based on XPath [18]. Whereas XSLT, XML-QL, and Quilt/XQuery use XML patterns for generating output (with the consequence that their output can only generate XML, but it cannot be used for manipulating an existing XML instance), our language deviates from these approaches: XPathLog works on an abstract, graph-based Herbrand-like model where an XPath-based syntax is used for querying (rule bodies) and generating/manipulating the data (rule heads). The semantics is given as sets of variable bindings; only when final output is produced by defining a result tree view, the well-known XML syntax/model is used.

Structure of the paper. The querying part of the language XPathLog is presented in Section 2 from the intuitive point of view. In Section 3, its theoretical foundation, XML Herbrand structures, are introduced. The semantics of query evaluation is described in Section 4, and Section 5 defines the semantics of XPathLog rules, focusing on the semantics of rule heads for generating and modifying XML data. Section 6 concludes the paper.

2 XPathLog

2.1 XPath: The Core Language
XPath [18] is the common language for addressing node sets in XML documents. It is based on navigation through the XML tree by path expressions of the form

```
//step/step/...step
```

Every single step is of the form

```
axis::nodetest[filter]
```

denoting that navigation goes along the given axis (i.e., child, descendant, parent, ancestor, preceding/following sibling, or attribute) in the XML document. Along the chosen axis, all elements which satisfy the nodetest (in most cases, the nodetest specifies an element type which subelements should be considered) are selected. From these, the ones which satisfy the given filter – roughly speaking, a predicate over XPath expressions which state conditions on a node (for details, see [18], or the formal semantics given in [16]) – are selected. Starting with this (local) result set, the next step is applied.

The most frequently used axes are abbreviated as

```
path/nodetest
```

for

```
path/child::nodetest
```

and

```
path//@nodetest
```

for

```
path/attribute::nodetest
```

Example 1 (XPath) Consider the following excerpt of the Mondial database [2] for illustrations.

```
<!ELEMENT mondial (country+, organization+, ...)>
<!ELEMENT country (name, city+, ...)>
<!ATTLIST country car . . .>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name, population*)>
<!ATTLIST city . . .>
<!ELEMENT organization (name, abbrev, members*)>
```

Proceedings of the International Database Engineering and Applications Symposium (IDEAS’01)
1098-8068/01 $10.00 © 2001 IEEE
XPath-Logic combines first-order logic and reference expressions. In this paper, we consider only the Horn fragment of XPath-Logic, i.e., logical rules over reference expressions and predicates over them (for full XPath-Logic, see [12]). In the remainder of the section, intuitive examples for XPathLog reference expressions are given.

Example 2 (Reference Expressions) Pure XPath expressions: pure XPath expressions (i.e., without variables) are interpreted as existential queries:

?- //country[name/text()="Germany"]//city/name. true

since the country element which has a name element with the text contents “Germany” contains at least one city descendant with a name element.

Output Result Set: The query “?- xpath→N” for any xpath binds N to all nodes belonging to the result set of xpath. For a result set consisting of elements, logical ids are returned:


• XPath-Logic LocationPaths may begin with constants or variables:

[2b] ConstLocPath ::= constant "/" RelLocPath | variable "/" RelLocPath

Definition 2 (XPathLog Atoms) Atoms are the basic components of XPathLog rules:

• an XPathLog atom is either a LocationPath or a ConstLocPath, or a predicate expression over these.

• an XPathLog atom is definite if it uses only the child and sibling axes and the atom does not contain negation, disjunction, function applications, and proximity position predicates (i.e., does not use the position() and last() functions). These atoms are allowed in rule heads (see Section 5.1); the excluded features would cause ambiguities what update is intended.

• an XPathLog literal is an atom or a negated atom,

• an XPathLog query is a list ?–L₁,…,Lₙ of literals (in general, containing free variables),

• an XPathLog rule is a formula of the form

A₁,…,Aₖ ← L₁,…,Lₙ

where Lᵢ are literals and Aᵢ are definite atoms.

Note that in contrast to usual Logic Programming, we allow for lists of atoms in the rule head which are interpreted as conjunctions.

XPath-Logic combines first-order logic and reference expressions. In this paper, we consider only the Horn fragment of XPath-Logic, i.e., logical rules over reference expressions and predicates over them (for full XPath-Logic, see [12]). In the remainder of the section, intuitive examples for XPathLog reference expressions are given.

Example 2 (Reference Expressions) Pure XPath expressions: pure XPath expressions (i.e., without variables) are interpreted as existential queries:

?- //country[name/text()="Germany"]//city/name. true

since the country element which has a name element with the text contents “Germany” contains at least one city descendant with a name element.

Output Result Set: The query “?- xpath→N” for any xpath binds N to all nodes belonging to the result set of xpath. For a result set consisting of elements, logical ids are returned:
Additional Variables: XPathLog allows to bind all nodes which are considered by an expression (both by the access path to the result set, and in the filters): The following expression returns all tuples \((N1, C, N2)\) such that the city with name \(N2\) belongs to the country with name \(N1\) and car code \(C\):

- \(\text{city} @ \text{name/text()} - C @ \text{car_code} \rightarrow N1\)
- \(\text{country}[\text{name/text()} = \text{“Germany”}]@\text{city} \rightarrow C\)

Dereferencing: For all organizations, give the name of the seat city and all names and types of members:

- \(\text{organization}@\text{name/text()} \rightarrow A\) and @seat/@name/text() \rightarrow S
- /members[@type=\text{T}]/@country/name/text() \rightarrow N

One element of the answer set is e.g.,

- A/\text{UN}/S/\text{New York}/T/\text{member}/N/\text{Germany}

Navigation Variables: Are there elements which have a name subelement with the PCDATA contents “Monaco”, and of which type are they?

- /\text{country}[\text{name/text()} = \text{“Monaco”}].
- Type/country \ X/country-monaco
- Type/city \ X/city-monaco

Schema Querying: The use of variables at name positions further allows schema querying, e.g., to give all names of subelements of elements of type \text{city}:

- /\text{city}/N.
- N/name
- N/population

Schema querying can e.g. be used for validation wrt. a DTD or a given XMLSchema (which can be present as an XML tree in the same XPathLog database).

Further examples can be found and executed with the LoPiX system [1].

3 XML Instances as Herbrand Structures

Logic-oriented frameworks are in general based on a semantical structure from which a suitable Herbrand-style structure is derived. The full theory can be found in [12]; here we directly define the Herbrand structure, following the DOM idea for representation of an XML instance.

The main features of the model are:

- the universe consists of the element nodes of the XML instance and literals used as attribute values and text contents;
- elements have properties, defined by (i) subelements (which are ordered) and (ii) attributes (which are unordered);
- multivalued attributes (NMtOKENS and IDREFS) are silently split;
- reference attributes are silently resolved.

Definition 3 (DOM Herbrand Structure) For a given XML instance, the Herbrand universe consists of

- \(\mathcal{N}\), a set of names (i.e., element and attribute names),
- \(\mathcal{V}\), a set of nodes, and
- \(\mathcal{L}\), a set of literals.

A DOM Herbrand structure \(\mathcal{H}\) over a given Herbrand universe consists of

- a set \(\text{pred}\(\mathcal{H}\)\) of predicate atoms over a set of predicate symbols and the Herbrand universe, and
- a (partial) mapping which associates with every \(x \in \mathcal{V}\) two lists of ground pairs:
  - \(A_H(\text{child}, x) \in ((\mathcal{V} \cup \mathcal{L}) \times \mathcal{N})^P\) and
  - \(A_H(\text{attribute}, x) \in ((\mathcal{V} \cup \mathcal{L}) \times \mathcal{N})^N\)

(recall that reference attributes are resolved silently into references to nodes).

For every pair \((v, n) \in A_H(\text{axis}, x)\), let \((v, n)\downarrow_1 := v\) and \((v, n)\downarrow_2 := n\).

Note that \(\mathcal{E}\) and \(\mathcal{A}\) are not direct interpretations of a signature \(\Sigma\), but mappings that “interpret” \(\mathcal{N}\), making attribute and element names full citizens of the language.

E.g., \(A_H(\text{child}, x) = ((v_1, n_1), (v_1, n_2), \ldots)\) denotes that the first child of \(x\) is \(v_1\) and it is of element type \(n_1\), etc.

There is a canonical mapping from the set of XML instances to the set of DOM Herbrand structures. The DOM Herbrand structure contains only the basic facts about the XML tree. For the other axes, \(A_H(\text{axis}, x)\) is derived from \(A_H(\text{child}, x)\) according to the XML specification.

Each node \(n\) of the XML Herbrand model is a potential root element for an XML tree view which recursively consists of all \(n\)'s subelements and attributes. The subtree rooted at an element \(r\) is a “useful” XML instance if

- the subelement relation in the subtree is acyclic, and
- the targets of outgoing references from the subtree also belong to the subtree.

Data Model. Although it “looks like” DOM, the data model significantly differs from the DOM and XML Query Data Model [17]: it allows for elements having multiple parents — i.e., updates may link elements as subelements of other elements which is a crucial feature for data restructuring and integration.

4 Evaluation of Queries

The semantics of XPathLog queries wrt. a DOM Herbrand structure is defined by structural induction by operators \(S\).
and $Q$ derived from the formal semantics given in [16]:

$$S_{H} : (\text{RefExprs} \times \text{Bindings}) \rightarrow \text{AnnotatedResults}^N$$

$$(\text{Axes} \times \mathcal{V} \times \text{RefExprs} \times \text{Bindings}) \rightarrow \text{AnnotatedResults}^N$$

evaluates reference expressions wrt. an axis, an (optional) context node and a given set of variable bindings and returns an annotated result list, i.e.,

- a result list, and
- with every element of the result list, a list of variable bindings (answers) is associated.

$$Q_{H} : (\text{PredExprs} \times \mathcal{V} \times \text{Bindings}) \rightarrow \text{Bindings}$$

evaluates filter expressions wrt. a context node and returns a set of variable bindings.

Negation in filters is interpreted as a “minus” operator, excluding some bindings from a given set of potential results. Thus, safety requirements apply similar to Datalog. The formal definition of the semantics by structural induction can be found in [12]. Here, we restrict ourselves to an example:

**Example 3 (Semantics)** Let $H$ be the canonical DOM Herbrand structure to the XML instance given in Example 1, let
eq expr := //organization→O[member/@country
[@car_code→C and name/text()→N]]
/abbrev/text()→A.

$$S_{H}(expr) =$$

list("	ext{UN}" , {O/un, A/"UN", C/"AL", N/"Albania"},
O/un, A/"UN", C/"GR", N/"Greece"),

("EU", {O/eu, A/"EU", C/"D", N/"Germany"},
O/eu, A/"EU", C/"F", N/"France"),

"	ext{EU}" , {O/eu, A/"EU", C/"D", N/"Germany"},
O/eu, A/"EU", C/"F", N/"France"),

Thus, searching for the 7th element of the above result set with

$$S_{H}(/[\text{organization}→\text{O}][...][7]/\text{abbrev/text()→A})$$

yields the name of the 7th organization, together with a set of variable bindings for the other variables. This is much different from defining a semantics yielding simply a list of variable bindings $O,..., A,..., C,..., N/...$ which would presumably yield the variable bindings for the 7th member of the UN.

5 Semantics of Rules

**Right Hand Side.** The body of an XPathLog rule is a set of XPathLog expressions. The evaluation of the body wrt. a given structure yields variable bindings which are propagated to the rule head where facts are added to the model.

5.1 Left Hand Side

Using logical expressions for specifying an update is perhaps the most important difference to approaches like XSLT, XML-QL, or Quilt/XQuery where the structure to be generated is always specified by XML patterns (this implies that these languages do not allow for updating existing nodes – e.g., adding children or attributes –, but only for generating complete nodes). In contrast, in XPathLog, existing nodes are communicated via variables to the head, where they are modified when appearing at host position of atoms.

The head of an XPathLog rule is a set of definite XPathLog atoms (cf. Definition 2). When used in the head, the “/” operator and the “[...]” construct specify which properties should be added or updated (thus, “[...]” does not act as a filter, but as a constructor). Recall that for the left hand side, proximity position predicates are not allowed; instead the position where a child or sibling should be inserted can be specified by

$$\text{host}[axis(i) \models \text{name} \rightarrow \text{value}]$$

where axis is either child or a sibling axis. If no position is given, the new element is appended at the end of the axis.

**Creation of Elements.** Elements can either be created as free elements by atoms of the form $\text{name}[...]$ (meaning “some element of type name” – in the rule head, this is interpreted to create an element which is not a subelement of any other element), or as subelements.

**Example 4** We create a new (free) country element with some properties:

```xml
/\text{country}[\text{@car_code}→\text{“BAV”} \text{ and } \text{@capital}→\text{X} \text{ and } \text{city}→\text{X} \text{ and } \text{city}→\text{Y}] : =
\text{city}→\text{X}[\text{name/text()}=\text{“Munich”}],
\text{city}→\text{Y}[\text{name/text()}=\text{“Nurnberg”}].
```

Note that the two city elements are linked as subelements. This operation has no equivalent in the “classical” XML model: these elements are now children of two country elements. Copying elements is described below. Thus, changing the elements effects both trees. By linking elements, it is possible to introduce cycles in the document hierarchy.

**Modification of Elements.** Already existing elements can be assigned as subelements to existing elements by using filter syntax in the rule head. When using the child or attribute axis for updates, the host of the expression gives the element to be updated or extended; when a sibling axis is used, effectively the parent of the host is extended with a new subelement. A ground instantiated atom

$$n[\text{child}::s→m]$$

makes $m$ a subelement of type $s$ of $n$; if the atom is of the form $n[\text{child}@i::s→m]$ or $n[\text{following-sibling}@j::s→m]$, this means that the new element to be inserted should be made the $i$th subelement
of \( n \) or \( j \) th sibling of \( n \). A ground instantiated atom of the form \( n[@a\rightarrow v] \) specifies that the attribute \( @a \) of the node \( n \) should be set or extended with \( v \). If \( v \) is not a literal value but a node, a reference to \( v \) is stored.

**Example 5** The following two rules are equivalent to the above ones:
\[
/country[@car_code=“BAV”].
\]
\[
C[@capital\rightarrow X and city\rightarrow Y] :-
//city\rightarrow X[name/text()=“Munich”], //city\rightarrow Y[name/text()=“Nurnberg”], //country\rightarrow C[@car_code=“BAV”].
\]
Here, the first rule creates a free element, whereas the second rule uses the variable binding of \( C \) for inserting subelements and attributes.

**Generation of Elements by Path Expressions.** Additionally, subelements can be created by path expressions in the rule head which create nested elements which satisfy the given path expression.

**Example 6** Bavaria gets a \( (PCDATA) \) subelement name: \( C/name[text()=“Bavaria”] :-
\)/country\rightarrow C[@car_code=“BAV”].

**Using Navigation Variables for Restructuring.** For data restructuring and integration, the intuitiveness and declarativeness of a language gains much from variables ranging not only over data, but also over schema concepts (classically, relation and columns, as, e.g., in SchemaSQL [10]). Such features have already been used for HTML-based Web data integration with F-Logic [9, 11].

Extending the XPath wildcard concept, XPathLog also allows to have variables at name position. Thus, it allows for schema querying, and also for generating new structures dependent on the data contents of the original one. Here, the we define the semantics of XPathLog to cast strings into names when a variable is bound to string contents in the body, and occurs at name position in the head:

**Example 7** From the elements
\[
<water type=“river” name=“Mississippi”> ...
</water>
\]
<water type=“sea” name=“North Sea”> ...
</water>,
the rule
\[
/T[@name\rightarrow N] :- //water[@type=T and @name\rightarrow N] creates <river name=“Mississippi”/> and <sea name=“North Sea”/>.
\]
Attributes and contents are then transformed by separate rules which use \( @name \) for identification. Properties are copied by using variables at element name and attribute name position:
\[
X[@A\rightarrow V] :- //water[@type=T and @name\rightarrow N and @A\rightarrow V], /T\rightarrow X[@name\rightarrow N].
\]
\[
X[S\rightarrow V] :- //water[@type=T and @name\rightarrow N and S\rightarrow V], /T\rightarrow X[@name\rightarrow N].
\]

**Formal Semantics.** The global semantics of XPathLog programs is defined by bottom-up evaluation based on a \( T_P \) operator similar to Datalog. As defined above, the left hand side of XPathLog rules is a list of definite atoms. The function
\[
\text{atomize} : \text{XPathLogExpr} \rightarrow \text{2XP atLogAtoms}
\]
resolves definite XPathLog atoms into atoms of the form \( n[\text{axis}:e\rightarrow w] \) and predicates over constants and variables which immediately define the extensions to the underlying structure \( H \) (again, the formal definition of atomize and of the \( T_P \) operator which evaluates rules and extends \( H \) with the resulting atoms can be found in [12]).

**Example 8 (Atomization)**
\[
?- //organization\rightarrow O[name/text()\rightarrow ON; @seat = members/@country[name/text()\rightarrow CN] /@capital].
\]
is atomized into
\[
?- root[descendant::organization\rightarrow O], O[name=“ON”], ON[text()] = ON, O[@seat=“S”], O[members=“M”], M[@country=“C”], //C[country\rightarrow Cap], _S = _Cap, _C[child::name=“CN”], _CN/text() = CN.
\]
The atomization introduces local variables which occur only in the head of the rule, i.e., the result is not a valid Logic Programming rule. Here, we follow the semantics of PathLog [7] which is implemented in [11] for object creation. After the atomization, the resulting atoms are processed in an ordering such that the local variables are bound to the nodes/objects which are generated. Thus, the rules are in fact safe.

**Example 9** For the rule given in Example 6, the atomized version of the rule head is
\[
C[name\rightarrow N], N[text()=“Bavaria”] :-
\)/country\rightarrow C[@car_code=“BAV”].
\]
The body produces the variable binding \( C/bavaria. \) When the head is evaluated, first, the fact 
\[
root[child::country\rightarrow x_1] \]
is inserted, adding an (empty) country subelement \( x_1 \) to \( root \) and binding the local variable \( _N \) to \( x_1 \). Then, the second atom is evaluated, generating the text contents to \( x_1 \).

An XPathLog program is a declarative specification how to create a set of XML documents from one or more input documents. Its global semantics coincides with the usual understanding of a stepwise process. Additionally, an intuitive meaning of negation and other nonmonotonic features can be expressed by user-defined stratification.

**Data Integration.** The data model is especially tailored to data integration, allowing to re-link elements into multiple overlapping trees, fusing elements, and introducing synonyms for subelement and attribute names (see [12]).
6 Related Work and Conclusion

Other approaches to semi-structured data, especially focusing on semi-structured data as databases (in contrast to documents) are OEM [8], YATL [4], and F-Logic [9, 11], using “proprietary” semi-structured data models of the respective languages (in pre-XMIL times). With these, also Logic Programming style languages have been used for manipulating and integrating semi-structured data(bases).

Several XML querying and transformation languages have already been mentioned in the introduction. XQL is a restricted pure querying language based on XPath. XPath also provides the basis for the transformation language XSLT, which is – similar to XPathLog – rule-based, but following a functional idea. XML-QL [5] is another language XSLT, which is – similar to XPathLog – rule-based, and integrating semi-structured data(bases).

XPathLog can be used both for querying XML documents, a strong theoretical foundation of the language concept. The Logic Programming background provides rule based bottom-up programming is easily understood from the XML perspective. Especially the nature of semantics which causes severe problems with maintenance of reference attributes).

Conclusion. To our knowledge, XPathLog is the first declarative, native XML language which allows for view definition and updates. XPathLog is completely XPath-based, i.e., both the rule bodies and the rule heads use an extended XPath syntax, thereby defining an update semantics for XPath expressions. The close relationship with XPath ensures that its declarative semantics is well understood from the XML perspective. Especially the nature of rule based bottom-up programming is easily understandable for XSLT practitioners, providing even more functionality. The Logic Programming background provides a strong theoretical foundation of the language concept. XPathLog can be used both for querying XML documents, and for restructuring and integration of XML sources. It has been implemented in the LoPiX system [1].

References