Querying Spatiotemporal XML Using DataFoX *

Yi Chen Peter Revesz
Computer Science and Engineering Department
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
{ychen,revesz}@cse.unl.edu

Abstract

We describe DataFoX, which is a new query language for XML documents and extends Datalog with support for trees as the domain of the variables. We also introduce for DataFoX a layer algebra, which supports data heterogeneity at the language level, and several algebra-based evaluation techniques.

1. Introduction

XML is a standard data model for data representation and exchange on the Internet. Several XML-based languages can also encode geographic information and spatiotemporal data. For example, the Vector Markup Language (VML) can represent vector objects, the Geography Markup Language (GML) can represent coordinates of OpenGIS features, and the Parametric Rectangle Markup Language (PRML), which we introduce in this paper, can represent parametric rectangles[11, 2].

While XML is successful for data representation, the current XML query languages, including XQuery[4], Quilt[5], Lorel[1], XML-QL[6], and XPathLog[10], do not fully support querying geographic and spatiotemporal XML documents. Querying based on relational database systems[3, 9, 7, 13, 12] also does not support spatiotemporal queries. Previous XML query language proposals also have the following common problems:

- The method of querying XML using relational database systems includes both the translation of XML schemas into relational schemas and the translation of tree-structured XML data into relational tables. This method may loose some semantic information that is encoded in the tree structure. For example, a rectangle and a straight-line object in GML have identical structures. There is no automatic translation algorithm that can maintain the XML schema information.

- Tree algebras do not allow grammar heterogeneity of the XML data. For example, a single spatial object with a rectangle shape can also be encoded as the combination of two triangles. The structure and the content of the GML data that result from these different combinations are different and tree algebras may erroneously regard them as two different objects.

- Many XML standards provide a strict structure for part of the data. For example, spatial attribute data in GML are well structured. Unfortunately, most previously proposed XML query languages and XML algebras do not provide sufficient support for well-structured data in a semi-structured environment. For example, previously proposed XML query languages do not support spatiotemporal queries on GML although the spatiotemporal attributes are well structured in GML.

To overcome the above problems, we present a rule-based XML query language, *Datalog For XML (DataFoX)*, which combines the simplicity of Datalog with the support for spatiotemporal data in constraint databases[8, 11]. We also introduce a *Layer Algebra*, which is a novel extension of relational algebra for XML and provides the basis of query evaluation and optimization for XML queries. The primary challenges we address are: (i) how to identify and represent trees in the query language, and (ii) how to evaluate the tree-based query language.

1.1. The DataFoX System Architecture

The architecture of the DataFoX system is outlined in Figure 1. XML Data sources in different formats are wrapped into a uniform representation called *Layer Constraint Databases*, which are powerful for capturing the

*This research was supported in part by NSF grant EIA-0091530 and a Gallup Research Professorship.

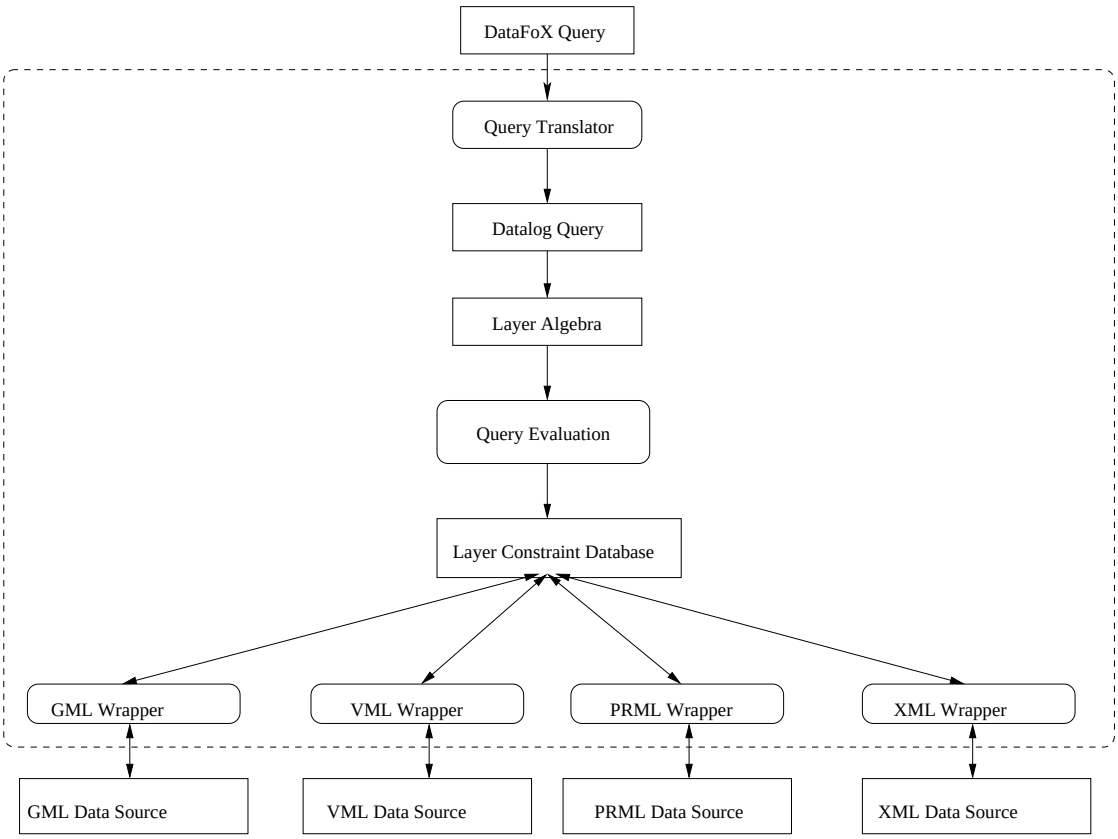


Figure 1. The DataFoX Architecture

XML tree structure and GML, VML, and PRML spatiotemporal information. The DataFoX query language is translated into a Datalog query, which is further translated into the layer algebra. The query is evaluated and the query result may be translated back to XML using wrappers.

Our architecture not only supports spatiotemporal queries but also enables integration of heterogeneous data sources. For example, Figure 2 shows an integrated university campus map, whose description is composed of three different data sources: the classroom buildings represented in GML, the stadium and the square in front of it represented in VML, and a moving bus and a helicopter represented in PRML.

The following are some typical queries on this map:

Query 1 Will the bus pass the east gate of the stadium?

Query 2 When will the helicopter fly over the stadium?

Query 3 When will the helicopter fly above the bus?

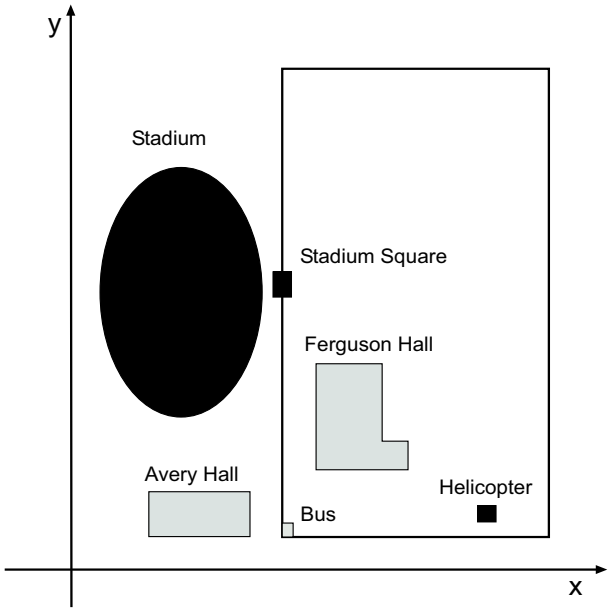
2. Data Model

The DataFoX system translates the different data sources (GML, VML, and PRML) into a layer model.

2.1. Layer Model

In XML, the type of each node is defined as an "element". We regard nodes as the counterpart of tuples in relational databases. The difference is that the data type of the field in a certain element could be another element. All of the nodes of the same element type belong to a *layer*, which is the counterpart of a relation in relational databases. The layer name is the same as the element name defined for these nodes. We say layer A is the *parent layer* of layer B if the element B is the sub-element of element A in the XML document.

We define *inferred layers* of a layer L as the set of layers, which are child layers of L. Similarly, the *inferred layer of a node* is the subtree that is rooted from this node. Thus, the inferred nodes of a certain node n and the edges between these nodes form an XML tree with root n. We assume that each node has a virtual attribute *nid* as the identifier of the node and a hidden attribute *pid* as a pointer to the parent node. We may use the *nid* to access a node, from which we may access its child nodes. In DataFoX, we also use this *nid* to identify the subtree that is rooted at the current node.



Campus map

```
<buildings>
  <lecture_hall>
    <name> Ferguson Hall </name>
    <dept> Computer Science </dept>
    <boundedby>
      <rectangle>
        <coord> <x> 35 </x> <y> 15 </y> </coord>
        <coord> <x> 45 </x> <y> 30 </y> </coord>
      </rectangle>
      <rectangle>
        <coord> <x> 45 </x> <y> 15 </y> </coord>
        <coord> <x> 50 </x> <y> 20 </y> </coord>
      </rectangle>
    </boundedby>
  </lecture_hall>
  .
  .
  .
</buildings>
```

GML document

```
<sport_facilities>
  <stadium>
    <name> Husker Stadium </name>
    <shape>
      <ellipse>
        <x> 15 </x>
        <y> 40 </y>
        <w> 10 </w>
        <h> 15 </h>
      </ellipse>
    </shape>
  </stadium>
  <square>
    <name> Stadium Square </name>
    <shape>
      <rectangle>
        <x> 30 </x>
        <y> 40 </y>
        <w> 5 </w>
        <h> 5 </h>
      </rectangle>
    </shape>
  </square>
</sport_facilities>
```

VML document

```
<transportation>
  <vehicle>
    <name> Bus </name>
    <schedule>
      <prectangle>
        <x>
          <from> <a> 0 </a> <b> 32 </b> </from>
          <to> <a> 0 </a> <b> 33 </b> </to>
        </x>
        <y>
          <from> <a> 1 </a> <b> 5 </b> </from>
          <to> <a> 1 </a> <b> 6 </b> </to>
        </y>
        <t>
          <from> 0 </from>
          <to> 55 </to>
        </t>
      </prectangle>
      .
      .
      .
    </schedule>
  </vehicle>
  .
  .
  .
</transportation>
```

PRML document

Figure 2. A campus map integrated from GML, VML, and PRML documents

We also assume *nid* is ordered, to maintain the order information of the XML tree model. For those elements defined as primitive data types, we assign an internal attribute named *data*. The value of this attribute is the content of the element.

We apply the same notation used to represent relational schema to represent the layer schema. However, the domain of variables for the layer schema is "tree". For example, the layer schema LAYERNAME($L_1, \ldots, L_n$) means that there is an element named "LAYERNAME" defined in the XML document, which has n child elements, with the layer names L_1, L_2, ..., L_n respectively. The data type of the child layers can be atomic (e.g.: string, integer etc.) or a tree type defined by another layer schema. An instance of a layer and all of its inferred layers is a tree, whose root is an instance of the root layer.

The layer definition can be generated from the element definition of the XML document. We assume that every XML document comes with a Data Type Definition (DTD). XML Schema is a stronger schema definition of XML, but DTD is adequate (note: we can always generate DTD from the XML schema to create the layer data model.) With the layer data model, an XML document is treated as a collection of layers, and the traversal of the XML document is actually the traversal between the layers.

```
<!ELEMENT library (book*)>
<!ELEMENT book    (title, year,
                   author+)>
<!ELEMENT title   #PCDATA>
<!ELEMENT year    #PCDATA>
<!ELEMENT author  #PCDATA>

<library>
   <book>
       <title>Landscape</title>
       <year>1997</year>
       <author> Steve </author>
   </book>
   <book>
       <title>Portrait</title>
       <year>2000</year>
       <author>John</author>
       <author>Maggie</author>
   </book>
</library>
```

Figure 3. Library Document

Example 2.1 (Layer Model) The XML document shown in Figure 3 can be modeled in the layer model shown in Figure 4. Every layer corresponds to an "element" definition in DTD. The *book* layer and *author* layer are called the *inferred layers* of the *library* layer. The layers with primitive data types like "PCDATA" are merged into their parent layer for simplicity.

2.2. Spatiotemporal Data Model

With the self-description feature of XML, every spatiotemporal data can be easily encoded in XML. Typically, GML defines a class of spatial features to represent OpenGIS data, and VML encodes vector information in XML. Parametric Rectangle data model, used to model moving objects, can be also encoded in XML, which we refer to as PRML.

These data models share a common characteristic, namely, the structures of the spatiotemporal attribute are well-formatted, and generally form a well-formatted subtree in the document (although the structure could be very complex.) Applying the layer data model, we may use the upper layer to *aggregate* the object using constraints. The wrapper performs this work. Figure 5 shows the aggregation operation on spatiotemporal data.

3. The Operators

A layer in XML can be regarded as a relation in relational databases that accept tree as the variable domain, which is defined as child layers. This definition allows us to define the operators on layers, which is very similar to that of relational databases. In this section, we introduce the Layer Algebra.

3.1. Selection

By "selection" operation, we are interested in all nodes in a layer, which satisfy some selection predicate. It is a "horizontal" operation on XML in the sense that no traverse between layers is involved. The input of selection in layer algebra is a layer L, and a set of predicates SL as parameters. It returns an output layer O, which has the same schema as the input layer L. The children of the output layer are lost except for those fields with atomic data types. The selection can be denoted as $\sigma^L_{SL}(L)$. A node n belongs to the output layers if it satisfies the selection predicate S.

3.2. Projection

We define the projection operation as an orthogonal operation to the selection operation. It is a "vertical" operation on XML in the sense that the traverse between layers is involved in the operation. The input of projection is a layer L (and all of its inferred layers), and a set of projection list PL as projection parameters. The projection list is a set of

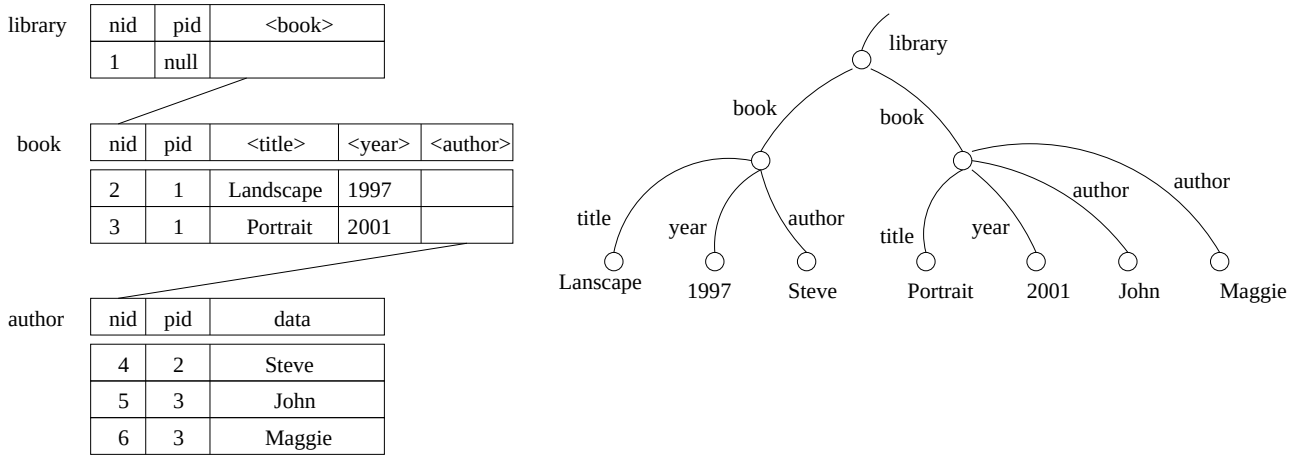


Figure 4. Layer Model for Library Document

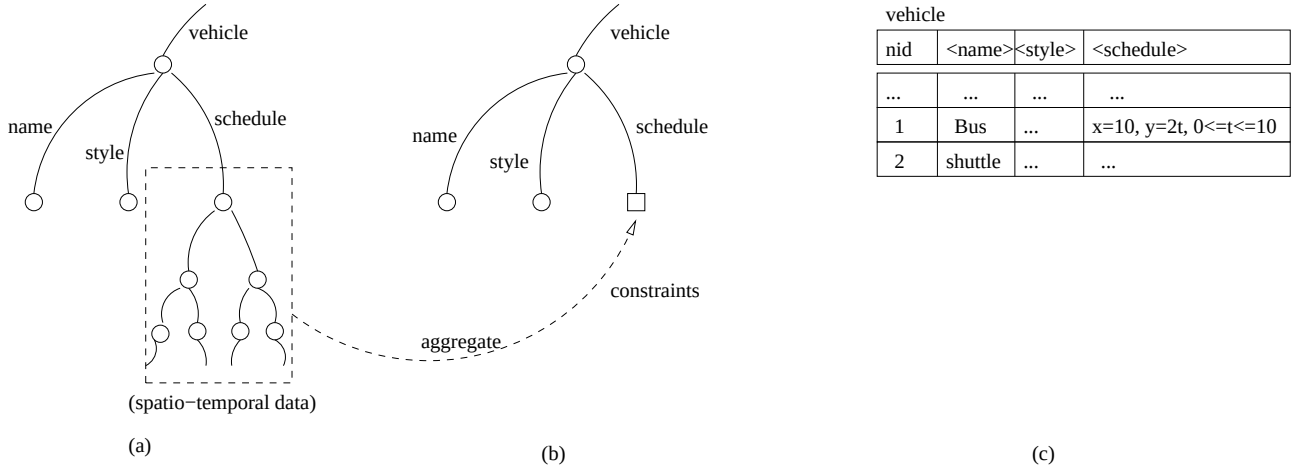


Figure 5. Spatiotemporal XML Aggregation

child layer names of the input layer L, which are associated with a set of sub-elements of the element defined for the input layer L. The output of projection operation is an output layer O, such that only those child layers that satisfy the projection predicates are kept as the inferred layer of output layer O. That is, the child layers in the output layer are a subset of child layers in the input layer, and all child layers of these returned layers are copied to the output. The projection can be denoted as $\pi^L_{PL}(L)$. Formally, the projection operation is defined as follows.

- A node in the input layer belongs to the output if and only if its element type is included in the projection list PL.

- A node in the input layers belongs to the output layers, if its the parent node belongs to the output layers.

3.3. Product and Join

The input of a product operation is two layers, L_1 and L_2, with the schema $L_1(L_{1,1}, ..., L_{1,m})$ and $L_2(L_{2,1}, ..., L_{2,n})$. The output of the product operation is a new layer O, with the layer schema $O(L_{1,1}, ..., L_{1,m}, L_{2,1}, ..., L_{2,n})$. Formally, the product operation can be defined as follows.

For every pair of nodes n_1 and n_2, such that n_1 belongs to layer L_1 and n_2 belongs to layer L_2, a new node n is created for the output layer O. The new nodes have $m + n$ children, such that the first m children are the same as the children of n_1, and the remaining n children are the same as the children of n_2.

The join operation can be expressed as selection operation posed on the result layer of the product operation. The formal definition of the join operation can be defined as follows.

Definition 3.1 (Join) Let L_1 and L_2 be two layers, with the schema $L_1(x_1, ..., x_i, ..., x_m)$ and $L_2(y_1, ..., y_j, ...y_n)$, respectively. Then the *join operation* $\bowtie$ creates a new layer L, such that

- The layer L has the schema $L(x_1, ..., x_i, ..., x_m, y_1, ..., y_j, ..., y_n)$.
- If a node n_1 belongs to layer L, there must be two nodes n_1 and n_2 that belongs to layer L_1 and L_2 respectively, such that the inferred layer x_i of node n_1 has the same value with that of the inferred layer y_j of node n_2.

The join operation can be denoted as $L_1 \bowtie^L_{x_i=y_j} L_2$.

3.3.1 Path Join

We introduce *Path Join* to join two layers with path predicates. The input is two layers A and B, and the output of the path join operation is two layers, A' and B', which has the same schema with A and B respectively. The path join operation will have the form of $A \bowtie^L_{PP} B$, while PP is the path predicates. The path join can be defined formally as follows.

- A node n belongs to the output layer of A' if and only if n also belongs to A, and there exists a node m in B', such that n and m satisfy the path predicate $n\theta m$.
- A node m belongs to the output layer of B' if and only if m also belongs to B, and there exists a node n in output layer A', such that m and n satisfy the path predicate $n\theta m$.

Example 3.1 The query *Find all information about books that were published in 1997* can be expressed using path join as follows:

$$(\sigma^L_{year=1997} book) \bowtie^L_{book.nid/author.nid} author$$

3.4. Translation of Layer Algebra

Layer Algebra is the basis for query evaluation and optimization. We show in this section the translation of Layer Algebra to Relational Algebra. With this translation we may implement the queries in a relational database system discussed in Section 4.

Theorem 3.1 Let L be an expression of Layer Algebra. There is an expression E in Relational Algebra that is equivalent to L.

proof: A Layer Algebra expression L that consists of projection, selection and join can be translated into a Relation Algebra expression E because there are equivalent operators for each layer algebra operator.

- (Selection) Selection operation only returns a set of nodes that belong to the input layer. Thus, the layer algebra $\sigma^L_{SL}(L)$ can be translated into relational algebra expression $\sigma_{SL}(R)$, such that the relation R has the same schema with the layer L.
- (Projection) For a projection operation with the form $\pi^L_{PL}(L)$, such that the PL is the list of fields to be projected, translate the expression into relational operation $\pi_{PL}(L)$, followed by $\sigma_{IL}(L)$, in which IL is the inferred layers of L.
- (Join) The join operation on two layers can be translated straightforwardly into a join operation on two relations.

For the selection, projection and join operation on a layer that do not have inferred layers, the translation is straightforward.

Example 3.2 Consider the join operation between the book layer and the publisher layer, $book \bowtie^L_{book.title=publish.title} publish$. Parts (a) and (b) in Figure 6 show the original data. Part (c) shows the schema of the output of the join operation. Finally, part (d) shows the tree structure of the result of the join operation.

4. DataFoX Language

DataFoX is a Datalog-like declarative query language. It is a high-level language that extends Datalog with tree type variables. We show that a DataFoX query can be translated into a constraint query, which can be evaluated in the running constraint database system MLPQ.

4.1. Syntax

The input of a DataFoX query is a set of XML documents, and the output of a DataFoX query is also an XML document. Each DataFoX query consists of a finite set of rules of the form:

$$\begin{aligned} Q(y_1, \ldots, y_m) \quad :- \quad & R_1(x_{1,1}, \ldots, x_{1,k_1}), \\ & \vdots, \\ & R_n(x_{n,1}, \ldots, x_{n,k_n}). \end{aligned}$$

where each R_i is either a name of element in the XML document or a defined relation name, including predefined spatiotemporal function names. The x's and y's are either variables, or constants. When $x_{i,j}$ is a variable, it is bounded to the jth child of the R_i element. We also allow to specify the sub-element type of the variable $x_{i,j}$, by adding the element name $e_{i,j}$, with the form of $e_{i,j} : x_{i,j}$.

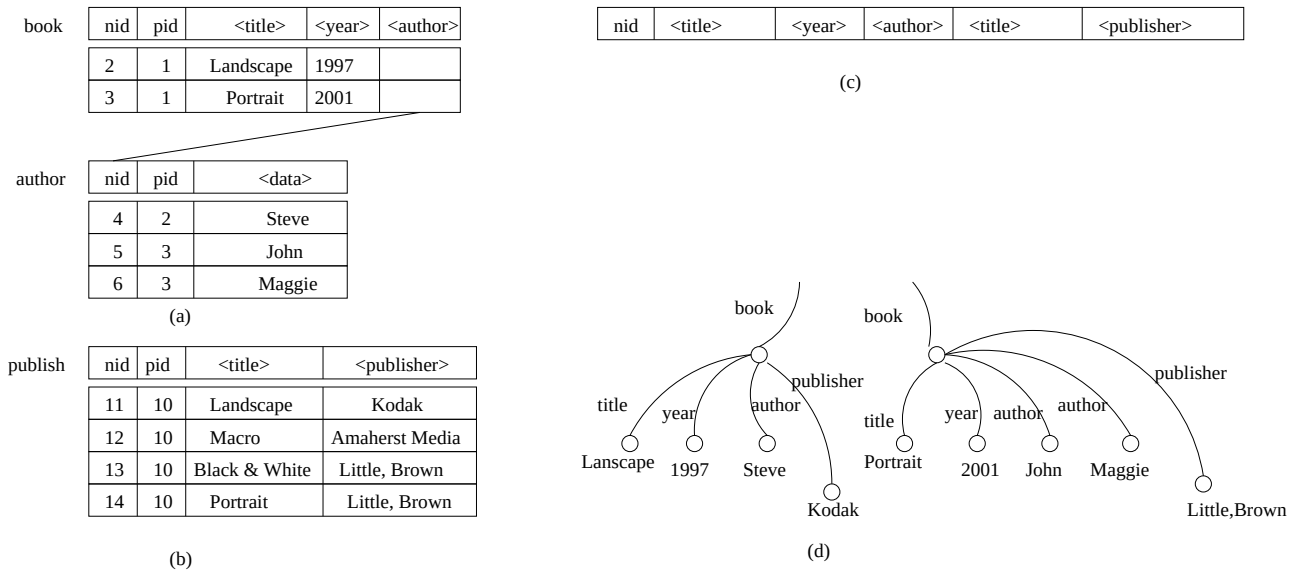


Figure 6. Join Operation

The rule head relation Q is the root element name of the query result. The variables y_i define the values of the elements which are the sub-elements of the resulting root element Q. Each variable y_i has to appear somewhere in the rule body.

4.1.1 DataFoX Query Body

We categorize the predicates in the DataFoX query body into three classes: Extensional and Intensional Predicates, Built-in Predicates and user defined functions. The same with Datalog, the extensional predicates are relational database relations and the intensional predicates are the database relations defined by the rules.

Built-in predicates include arithmetic comparison predicates, =, $<$, and so on. We introduce two path predicates, "/" and "//" refer to the parent-child and ascendant-descendant relationships of two nodes. For example, x/y means the node x is the parent node of the y.

Tree is included in the domain of variables in DataFoX. An internal attribute node ID (nid) for an extensional predicate refers to the current node that satisfies the predicate, and this nid is used to identify the tree with the root node nid. For example, in the predicate $A(a, B : b, C : c)$ the variable a refers to the current node with the element type A, which has two sub-elements B and C. Two boolean predicates will hold from this extensional predicate: a/b and a/c, because the node b and c are both children of node a.

We introduce user defined predicates to handle the operation between tree type variables. Tree data is more complex then atomic data. The predicate between two tree variables can't simply be mapped to the predicates of the nodes and edges of the tree. For example, a single spatial object can be divided into two rectangles in two different ways. Thus, we can encode this spatial object in GML in two different ways but they are describing exactly the same object. Equality, as applied to two trees, means that there exists a mapping between nodes or edges of the two trees. This concept focuses on the equality of structure detail, but misses the semantic information provided by the tree as a whole.

The introduction of user defined predicates also allows hiding of structure detail of variable from user. This provides a data integration mechanism.

Example 4.1 Consider the query:
Find all the buses that will intersect with the bus named "schoolbus" between times 0 and 100.
This can be expressed as:

```
vehicle(bus: u):-
        Bus(v, name:"schoolbus"),
        Bus(u),
        intersect(v, u, x, y, t),
        0<=t, t<=100.
```

In this example, bus u and v have the same data type, defined as "Bus" element in the document, but the structure details of the data are hidden. The user defined predicate intersect handle the detail and heterogeneity of these two trees.

4.1.2 DataFoX Query Head

Each DataFoX query head has two roles: (1) it specifies the projection operation on the relation created by the query body, and (2) it defines the schema for each variable.

This means that each variable in the DataFoX rule head must appear in the rule body. The data type of a variable in the rule head is the same as that of the matching variable in the rule body. The query result is tagged as an XML document according to the query head.

4.2. Tree Operation

One of the advantages of Layer Algebra for XML is that it provides a data abstraction based on the tree data types. The detail of a subtree in XML is abstracted as a tree type variable in the root layer of this subtree. With this tree abstraction we may design operations between trees. The details of the tree variable (the structure and content) are all hidden from the user. In implementation, the corresponding XML wrapper will be triggered when a tree variable is used in the query.

A typical application of tree operation is the manipulation of spatiotemporal data encoded in XML.

Example 4.2 Consider the query:
Find the intersection time and position of two vehicles.
This can be expressed as:

```
intersect_time(x, y, t) :-
    vehicle(name:"Bus", schedule:s1),
    vehicle(name:"Shuttle",schedule:s2),
    intersect(s1, s2, x, y, t).
```

The schedule layer and its inferred layers encode spatiotemporal information of the vehicle in a tree structure. Tree variables $s1$ and $s2$ are used to abstract the tree and take part in the intersect spatiotemporal operation. The tree structure details are hidden from the user.

This kind of query is very difficult to express in other XML query languages. Consider the intersection of two line segments AB and CD represented in GML. The coordinates of the four points can be queried by XQuery. Suppose these four points are represented in coordinates $A(0,0)$, $B(5,5)$, $C(0,5)$, $D(5,0)$. The coordinates of the intersection point is $(2.5, 2.5)$, which does not exist in the XML source. To handle spatial data types is an even more complex problem. For example, the intersection of two rectangular objects could be another rectangle, but it also could be a line segment or a single point. To represent this query with XQuery language, the user has to take into account each case separately.

In the layer model we may use a tree type variable to represent the spatiotemporal object and introduce a set of built-in operations defined on these variables. For example, a function $area()$ can be defined on the Bounded by element in GML to compute the area of the region represented by this element. The task of translating the spatiotemporal objects to a constraint representation is shifted to the wrapper. That makes the query language easier to understand.

4.3. DataFoX Evaluation

DataFoX extends Datalog with path predicate and tree operation, which supports the tree data type domain. In this section, we address the problem of DataFoX evaluation.

Theorem 4.1 (Least Fixpoint Evaluation) The least fixpoint of any DataFoX query and input database (XML document) with path constraints is closed-form evaluable.

4.4. Translating DataFoX Queries

DataFoX uses a similar syntax to Datalog, and Layer Algebra is an extension of Relational Algebra with tree operations. In this section, we discuss the translation of DataFoX rule bodies into Layer Algebra.

Algorithm 1 Computing the layer for a DataFoX rule body using Layer Algebra
INPUT: The body of a DataFoX rule r, which consists of subgoals $S_1, ..., S_n$. For each $S_i = p_i(A_{i,1}, ..., A_{i,k_i})$ with an ordinary predicate, there is a layer L_i already computed where the A's are either an internal attribute nid for the layer or a variable term with the form $e_{i,k} : x_{i,k}$, or a constant term with the form $e_{i,k} : c_{i,k}$.
OUTPUT: An expression of layer algebra.
METHOD:

1. For each subgoal S_i, let Q_i be the expression $\pi^L_{PL}(\sigma^L_{SL}(L_i))$, such that all variables appear in the predicate are included in the projection term pl, and sl is the conjunction of the following conditions: (i) If there is a constant a appear in the subgoal with the form $e_{i,k} : a$, then sl has the term $e_{i,k} = a$; and (ii) if two sub-elements have the same variable with the form $e_{i,k} : x, e_{i,l} : x$, then sl has the term $e_{i,k} = e_{i,l}$.

2. For each subgoal S_i which is path predicate with the form $a\theta b$, let Q_i be the expression $L_{i,j} \bowtie_{L_{i,j}.nid\theta L_{i,k}.nid} L_{i,k}$, such that the variable a and b appears as the internal attributes in two other subgoals S_j and S_k respectively.

Theorem 4.2 Let Q be a DataFoX query not involving recursion, function calls (spatiotemporal functions), there is an expression E in layer algebra that is equivalent to Q.

4.5. Translation to Layer Algebra

With Algorithm 1 all spatiotemporal operations can be converted into constraint queries. In this section, we focus on the translation to layer algebra. We also show how to eliminate path predicates.

The query body of DataFoX can be translated into Datalog easily, provided a schema information of the XML document (DTD format).

To simplify the query statement, DataFoX extensional and intensional predicates allow the explicit specification of the field name. The complete predicate can be retrieved by looking up the internal schema information.

The translation of DataFoX to layer algebra can be summarized by the following steps:

1. For a predicate, if there is any constants appear in it,the predicate can be translated into a "selection" operation. For example, the predicate $book(title : "portrait", ...)$ can be translated into $\pi_{title='portrait'}(book)$.

2. For any variable that appears in more than one predicate, a "join" operation should be used between these predicates. For example, the DataFoX query Example 3.2 can be expressed as:

```
publishers():- book(title:t),
               publish(title:t).
```

   The same variable t appears in two predicates, the corresponding algebra is $book \bowtie^{L}_{book.title=publish.title} publish$.

3. The path predicate, for example, the predicate with the form a/b, can be regarded as a special path join operation between the two nodes and the edge relation. We can easily translate a path predicate into layer algebra. Note that by giving an edge relation that stores all edges, the descendant predicate with the form $a//b$ can also be translated into a set of join operations on the edge relation. The descendant predicate can be described as recursive Datalog as follows:

```
descendant(a, b) :- parent(a,b).
descendant(a, b) :- parent(a,c),
                    descendant(c,b).
```

In DataFoX evaluation, we maintain a lightweight tree to abstract the path information and use a bottom-up traversal to bypass the evaluation of recursive join operations.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[2] M. Cai, D. Keshwani, and P. Revesz. Parametric rectangles: A model for querying and animation of spatiotemporal databases". In *Proc. Seventh Conference on Extending Database Technology (EDBT), Springer-Verlag LNCS 1777*, pages 430–444, 2000.

[3] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB (Informal Proceedings)*, pages 105–110, 2000.

[4] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. Xquery: A query language for XML, 2001.

[5] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. In *WebDB (Informal Proceedings)*, pages 53–62, 2000.

[6] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML.

[7] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *SIGMOD 99*, pages 431–442, 1999.

[8] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Journal of Computer and System Sciences , vol. 51, no. 1*, pages 26–52, 1995.

[9] M. Fernandez and W. Tan and D. Suciu. SilkRoute: Trading between Relations and XML. In *WWW9*, May 2000.

[10] W. May. Xpath-logic and xpathlog: A logic-based approach for declarative xml data manipulation.

[11] P. Revesz. *Introduction to Constraint Databases*. Springer, 2002.

[12] J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton, and I. T. Rinov. A general technique for querying xml documents using a relational d atabase system. In *SIGMOD Record*, pages 20–26, 2001.

[13] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *The VLDB Journal*, pages 302–314, 1999.