Querying XML with Constraints

M.-S. Hacid E. Terzi
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA

A. Vakali
Department of Informatics
Aristotle University
54006 Thessaloniki - Greece

Abstract *XML is a language for the description of structured documents and data. It is on the way to become the new standard for data exchange, publishing, and developing intelligent Web agents. XML is based on the concept of documents composed of a series of entities (i.e., objects). Each entity can contain one or more logical elements. Each of these elements can have certain attributes (properties) that describe the way in which it is to be processed. XML provides a formal syntax for describing the relationships between the entities, elements and attributes that make up an XML document. In this paper, we introduce a framework for querying XML databases by specifying ordering constraints over documents.*

Keywords: Role Tress, Subsumption, Constraints, Rules, Query Languages.

1 Introduction

XML [1] is a new standard adopted by the World Wide Web Consortium (W3C) to complement HTML for data exchange on the Web. It is a data format for Web applications. XML documents do not have to be created and used with respect to a fixed, existing schema. This is particularly useful in Web applications, for simplifying exchange of documents and for dealing with semistructured data. Its emergence as a standard for data representation on the Web is expected greatly to facilitate the publication of electronic data by providing a simple syntax for data that is both human- and machine-readable.

In many respects, XML data is an instance of semistructured data [2]. XML documents comprise hierarchically nested collections of *elements*, where each element can be either atomic (i.e., raw character data) or composite (i.e., a sequence of nested subelements). Further, *tags* stored with elements in an XML document describe the semantics of the data rather than simply specifying how the element is to be displayed (as in HTML). Thus, XML data, like semistructured data, is hierarchically structured and self-describing.

In this paper, we propose, in the form of constraints, a notion of ordering to capture the relationship between XML documents. Intuitively, ordering captures the fact that one document is *smaller than* (or *subsumes*; or, *approximates*) another document, or the fact that one document is compatible with another. We also show how these constraints can be used to query XML databases.

Paper outline: Section 2 summarizes the contributions of this paper. In Section 3 we give our structure for representing XML documents, and the constraints we allow to reason about the documents. In Section 4, we develop our query language and give its syntax and semantics. We conclude in Section 5.

2 Contributions

We present a class of constraints, namely, ordering constraints, that are of interest in XML databases. Our constraints are inspired by Feature Logics [3, 4, 5]. Feature descriptions are used as the main data structure of socalled unification grammars, which are a popular family of declarative formalisms for processing natural language . They provide for a partial description of abstract objects by means of functional attributes called features. On top of this constraint language, we allow the definition of relations (by means of definite clauses) in the style of [6], leading to a declarative, rule-based, constraint query language for XML. The language we propose is based on the general scheme for handling clauses whose variables are constrained by an underlying constraint theory [7].

To summarize, the framework presented here integrates formalisms developed in Databases, Feature Logics and Constraint (Logic) Programming. The paper builds on the works by [4, 8, 9, 7] to propose a structure for XML data and a declarative, rule-based, constraint query language that has a clear declarative and operational semantics. We make the following contributions:

(1) We develop a simple and flexible structure for representing XML data. The structure, called *role trees*, is inspired by Feature Constraint Systems. Trees are useful for structuring data in modern applications. This gives the more flexible role trees (our data structure) an interesting potential.

(2) We propose a constraint language for XML data. The ordering constraints allow to declaratively specify relationships between trees representing XML data. Our constraints are of a finer grain and of different expressiveness.

(3) We propose a declarative, rule-based, constraint query language that can be used to infer relationships between XML data. We view our query language as consisting of a constraint language on top of which relations can be defined by definite clauses. The language has declarative and operational semantics.

The structure we use to represent XML documents is an adaptation of [10] where we introduce two labels, namely *subelement* and *value*, to name the edges of the tree.
The major improvement over existing XML query languages (e.g., XML-QL [11] , chapter 5) is the use of new constraints to constrain the retrieval of XML documents. To our knowledge, no previous work considers the kind of constraints we propose in this paper.
Although in the basic form that we give here, the formalism does not account for all aspects of XML data, it constitutes a kernel to be extended.

3 Data and Query Modeling

3.1 XML Data Representation

XML[1] is a textual representation of data. The basic component in XML is the *element*, that is, a piece of text bounded by matching tags such as < faculty > and < /faculty >. Inside an element we may have "raw" text, other elements, or a mixture of the two. Consider the following XML example:

```
  < faculty >
    < name > Clint < /name >
    < room > 420 < /room >
    < email > crm@cs.edu < /email >
< /faculty >
```

An expression such as < faculty > is called a start-tag and < /faculty > an end-tag. Start- and end-tags are also called markups. Such tags must be balanced; that is, they should be closed in inverse order to that in which they are opened, like parentheses. Tags in XML are defined by users; there are no predefined tags, as in HTML. The text between a start-tag and the corresponding end-tag, including the embedded tags, is called an *element*, and the structures between the tags are referred to as the *content*. The term *subelement* is also used to describe the relation between an element and its component elements.

[1]For more details, see [11].

Thus < email > ... < /email > is a subelement of < faculty > ... < /faculty > in the example above. As with semistructured data, we may use repeated elements with the same tag to represent collections. The following is an example in which several < faculty > tags occur next to each other.

```
 < people >
  < faculty >
      < name > Clint < /name >
      < room > 420 < /room >
      < email > crm@cs.edu < /email >
  < /faculty >
  < faculty >
      < name > Marion < /name >
      < room > 319 < /room >
      < email > mj@cs.edu < /email >
  < /faculty >
< /people >
```

The basic XML syntax is perfectly suited for describing semistructured data. Recall the syntax for semistructured data expressions. The simple XML document

```
 < faculty >
   < name > Clint < /name >
   < room > 420 < /room >
   < email > crm@cs.edu < /email >
< /faculty >
```

has the following representation as a semistructured data expression:

{faculty : {name : "Clint", room : 420,
                    email : "crm@cs.edu"}}

There is a subtle distinction between an XML element and a semistructured data expression. A semistructured data expression is a set of label/subtree pairs, while an element has just one top-level label. XML denotes graphs with labels on nodes[2]. In this paper, we consider that an XML document is represented by a role tree of a specific form. The only two edge labels are subelement and value. Figure 1 illustrates our representation for the XML data above.

[2]While semistructured data expressions denote graphs with labels on edges.

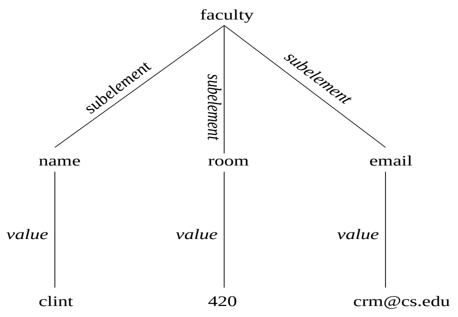

Figure 1: Our labeled-tree representation for XML data.

Path expressions describe path along the graph, and can be viewed as compositions of labels. For example, the expression

subelement.value

describes a path that starts in an object, continues to the subelement of that object, and ends in the value of that subelement.

3.2 XML Tree Structure

3.2.1 Role Tree Structure

The Graph-Oriented Model. Formally, XML data is represented by a directed labeled graph $G = (N, E)$ where N is a finite set of labeled nodes and E is a finite set of labeled edges. An edge e is written as (n_1, α, n_2) where n_1 and n_2 are members of N and α is the label of the edge.

In this paper, we use the notion of trees to represent XML data. We investigate a set of constraints over XML data. Before presenting the constraint language, we shortly discuss feature and role trees.

A *feature tree* is a tree with unordered, labeled edges and labeled nodes. The edge labels are called features; features are functional in that two features labeling edges departing from the same node are distinct. In programming, features correspond to record field selectors and node labels to record field contents. In our framework, we extend the notion of feature trees to role trees. A role tree is a possibly infinite tree whose nodes are labeled with symbols called sorts, and whose edges are labeled with symbols called roles. The labeling with roles is nondeterministic in that edges departing from a node need not to be labeled with distinct roles.

An example of a role tree is shown on figure 1. Its root is labeled with the node label faculty and the edge departing at this root is labeled by the role subelement.

A role tree is defined by a tree domain and a labeling function. The domain of a role tree τ is the multiset of all words labeling a branch from the root of τ to a node of τ. For instance, the domain of the tree of figure 1 is {ϵ, subelement, subelement, subelement, subelement.value, etc.}. The labeling function associates each element of the domain to a set of sorts.

A role tree is finite if its tree domain is finite. In general, the domain of a role tree may also be infinite in order to model XML data with cyclic dependencies.

A role tree can be seen as a carrier of information. This viewpoint gives rise to an ordering relation on role trees in a very natural way that we call *information ordering.* The information ordering is illustrated by the example of figure 2. The smaller tree is missing some information about the object it represents, namely that this object is a faculty and that role *subelement* of the object has an additional tag, *email*, whose value is *crm@cs.edu.* In order to have nodes without information, we allow for unlabeled nodes depicted with a •. Formally, this means that we do not require a labeling function to be total.

Intuitively, a role tree τ_1 is smaller than a role tree τ_2 if τ_1 has fewer edges and node labels than τ_2. More precisely, this means that every word of roles in the tree domain of τ_1 belongs to the tree domain of τ_2 and that the partial labeling function of τ_1 is contained in the labeling function of τ_2. In this case we write $\tau_1 \leq \tau_2$. The notions of tree domains and labeling function will be formally defined below.

The following are examples of queries. In these queries, x and y are tree variables (i.e., variables ranging over role trees), and α, β are path variables (i.e., variables ranging over composition of roles). We use the predicate symbol tree to denote the set of trees in the database. The formal semantics of the constructs used in constraints will be given later.

$$\mathsf{Answer}(\mathsf{x}) \leftarrow \mathsf{tree}(\mathsf{x}) \| \{\mathsf{John}\}(\alpha, \mathsf{x})$$

This query returns the set of trees such that there is a path (here the valuation of the variable α) from the root (of each tree answer to the query), leading to the sort (i.e., value) {John}. In this query (expressed as a rule), Answer(x) is called the head of the query and $\mathsf{tree}(\mathsf{x}) \| \{John\}(\alpha, x)$ is called the body of the query. The notation $\mathsf{S}(\alpha, \mathsf{x})$ means that there is a path α in the tree x leading from the root to the set of sorts S.

$$\mathsf{Answer}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{tree}(\mathsf{x}), \mathsf{tree}(\mathsf{y}) \| \mathsf{x} \leq \mathsf{y}, \{\mathsf{John}\}(\alpha, \mathsf{x}), \{\mathsf{John}\}(\beta, \mathsf{y})$$

This query returns a set of pairs (x, y) of trees such that x subsumes y and there is a path (i.e., valuation of α) in x and a path (i.e., valuation of β) in y leading to the same set of sorts (here {John}).

$$\mathsf{Answer}(\mathsf{x},\mathsf{y}) \leftarrow \mathsf{tree}(\mathsf{x}), \mathsf{tree}(\mathsf{y}) \| \mathsf{x} \sim \mathsf{y}$$

This query returns pairs of trees that are compatible. The symbol $\sim$ stands for compatibility.

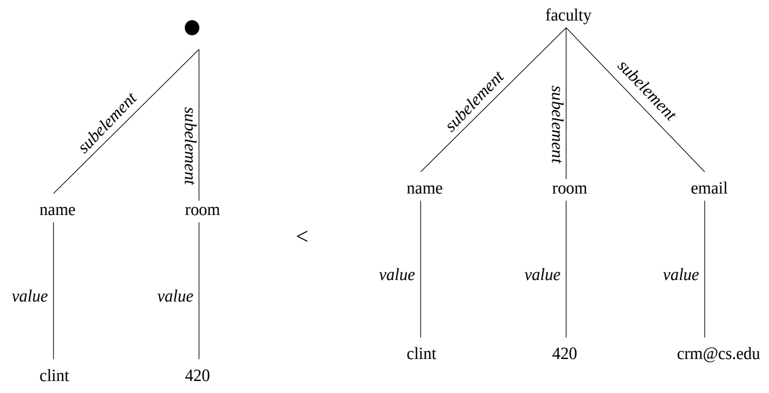

Figure 2: Example of an order over trees.

To give a rigorous formalization of role trees, we first fix two disjoint alphabets $\mathcal{S}$ and $\mathcal{F}$, whose symbols are called *sorts* and *roles*, respectively. The letters S, S' will always denote sets of sorts, and the letters f, g, h will always denote roles. Words over $\mathcal{F}$ are called paths. The concatenation of two paths v and w results in the path vw. The symbol ϵ denotes the empty path, $v\epsilon = \epsilon v = v$, and $\mathcal{F}^*$ denotes the set of all paths.

A *tree domain* is a nonempty set $D \subseteq \mathcal{F}^*$ that is prefix-closed; that is, if $vw \in D$, then $v \in D$. Thus, it always contains the empty path.

A *role tree* is a mapping $t : D \rightarrow \mathcal{P}(\mathcal{S})$ from a tree domain D into the powerset $\mathcal{P}(\mathcal{S})$ of $\mathcal{S}$. The paths in the domain of a role tree represent the nodes of the tree; the empty path represents its root. The letters s and t are used to denote role trees.

A *path* p is a finite sequence of roles in $\mathcal{F}$. The *empty path* is denoted by ϵ and the free-monoid concatenation of paths p and p' as pp'; we have $\epsilon p = p\epsilon = p$. Given paths p and p', p' is called a *prefix of* p if $p = p'p''$ for some path p''. A tree domain is a non-empty prefixed-closed set of paths.

Definition 1 (Role Trees). *A role tree τ is a pair (D, L) consisting of a tree domain D and a partial labeling function $L : D \rightarrow \mathcal{P}(\mathcal{S})$. Given a role tree τ, we write D_τ for its tree domain and L_τ for its labeling function. A role tree is called finite if its tree domain is finite, and infinite otherwise. We denote the set of all role trees by $\mathcal{R}$. If $p \in D_\tau$ we write as $\tau[p]$ the subtree of τ at path p which is formally defined by $D_{\tau[p]} = \{(p', S) \mid pp' \in D_\tau\}$ and $L_{\tau[p]} = \{(p', S) \mid (pp', S) \in L_\tau\}$.*

3.3 Constraints over Role Trees

3.3.1 Syntax and Semantics of Constraints

In the following, we introduce the syntax and semantics of ordering constraints over role trees. We assume an infinite set (which we denote by $\mathcal{V}$) of tree variables ranged over by x, y, an infinite set (which we denote by $\tilde{V}$) of path variables ranged over by α, β, an infinite set $\mathcal{F}$ of *roles*[3] ranged over by f, g, and an arbitrary multiset $\mathcal{S}$ of sorts denoted by S, T containing at least two distinct elements.

Syntax. An *ordering constraint* φ is defined by the following abstract syntax.

$$\varphi ::= x \leq y \mid S(\alpha, x) \mid x[f]y \mid x \sim y \mid \varphi_1 \wedge \varphi_2$$

An ordering constraint is a conjunction of *atomic constraints* which are either *atomic ordering constraints* $x \leq y$, *generalized labeling*

[3]In our case, a role is either *subelement* or *value*.

constraints $S(\alpha, x)$, *selection constraints* $x[f]y$, or *compatibility constraints* $x \sim y$.

Semantics. The signature of the structure contains the binary relation symbols $\leq$, $\sim$, and $S(\bullet,\bullet)$ (for every set of labels S), and for every role f a binary relation symbol $\bullet[f]\bullet$. The domain of the structure $\mathcal{R}$ is the set of possibly infinite role trees. The relation symbols are interpreted as follows:

$$\begin{array}{rll} \tau_1 \leq \tau_2 & \text{iff} & D_{\tau_1} \subseteq D_{\tau_2} \text{ and } L_{\tau_1} \subseteq L_{\tau_2} \\ \tau_1[f]\tau_2 & \text{iff} & D_{\tau_2} = \{p \mid fp \in D_{\tau_1}\} \text{ and} \\ & & L_{\tau_2} = \{(p, S) \mid (fp, S) \in L_{\tau_1}\} \\ S(\alpha, \tau) & \text{iff} & \mu(\alpha) \in D_\tau \text{ and } (\mu(\alpha), S) \in L_\tau \\ \tau_1 \sim \tau_2 & \text{iff} & L_{\tau_1} \cup L_{\tau_2} \text{ is a partial function} \\ & & \text{(on } D_{\tau_1} \cup D_{\tau_2}) \end{array}$$

where μ is a valuation from $\tilde{V}$ to the set of elements $\mathcal{F}^*$.

3.3.2 Satisfiability Test

We present a set of axioms valid for our constraint system and then interpret these axioms as an algorithm that solves the satisfiability problem of our constraint system.

Table 1 contains axioms schemes F1 - F6 that we regard as sets of axioms. The union of these sets of axioms is denoted by F. For instance, an axiom scheme $x \leq x$ represents the infinite set of axioms obtained by instantiation of the meta variable x. An axiom is either a constraint φ, an implication between constraints $\varphi \rightarrow \varphi'$, or an implication $\varphi \rightarrow false$.

The role tree structure $\mathcal{T}$ is defined as follows:

- The domain of $\mathcal{T}$ is the set of all role trees.
- $t \in A^{\mathcal{T}}$ if and only if $t(\epsilon) = A$ (t's root is labeled with the sort A).
- $(s, t) \in f^{\mathcal{T}}$ if and only if $f \in D_s$ and $t = fs$ (t is the subtree of s at f).

Proposition 1 *The structure $\mathcal{T}$ is a model of the axioms in* F.

4 Constraint-Based Query Language for XML

We now present a construction that, given the constraint language, let us call it $\mathcal{C}$ (for ordering constraints) and a set of relation symbols, extends $\mathcal{C}$ to a constraint query language $\mathcal{R}(\mathcal{C})$. Hence, we view our query language as consisting of a constraint language on top of which relations can be defined by definite clauses. The language has a declarative and operational semantics. Regarding the operational semantics, we show that the immediate consequence operator is monotonic and continuous. Hence, queries can be evaluated in a bottom-up iterative fashion.

Definition 2 (Predicate Symbol) *we define the predicate symbol* **tree** *to denote XML data represented as trees.*

We reason about XML data by a program P which contains a set of rules defining ordinary predicates. The rules are of the form:

$$H(\bar{X}) \leftarrow L_1(\bar{Y_1}), \ldots, L_n(\bar{Y_n}) \| c_1, \ldots, c_m$$

for some $n \geq 0$ and $m \geq 0$, where $\bar{X}, \bar{Y_1}, \ldots, \bar{Y_n}$ are tuples of tree variables or path variables. We require that the rules are safe, i.e., a variable that appears in $\bar{X}$ must also appear in $\bar{Y_1} \cup \ldots \cup \bar{Y_n} \cup$ { path variables and tree variables occurring in $c_1, \ldots, c_m$ }. The predicates $L_1, \ldots, L_n$ may be either *tree* or ordinary predicates. $c_1, \ldots, c_m$ are ordering constraints ($\mathcal{C}$-constraints). In the following, we use the term (positive) atom to make reference to predicates $L_1, \ldots, L_n$.

Our language has a declarative model-theoretic and fixpoint semantics.

5 Conclusion

There is a growing interest in XML. As XML data (e.g., on the Web) proliferate, aids to

F1.1	$x \leq x$
F1.2	$x \leq y \wedge y \leq z \rightarrow x \leq z$
F2	$x[f]x' \wedge x \leq y \wedge y[f]y' \rightarrow x' \leq y'$
F3.1	$x \sim x$
F3.2	$x \leq y \wedge y \sim z \rightarrow x \sim z$
F3.3	$x \sim y \rightarrow y \sim x$
F4	$x[f]x' \wedge x \sim y \wedge y[f]y' \rightarrow x' \sim y'$
F5	$S(\alpha, x) \wedge S'(\alpha, x) \rightarrow false$ for $S \neq S'$
F6	$S(\alpha, x) \wedge S'(\alpha, y) \wedge x \sim y \rightarrow false$ for $S \neq S'$

Table 1: Axioms of Satisfiability: F1-F6

browsing and filtering become increasingly important tools for interacting with such exponentially growing information resources and for dealing with access problems. In this paper we have shown how a class of constraints can lead to make a flexible query language for XML data. In this paper, sorts appearing in ordering constraints (i.e., constraints of the form $\mathsf{S}(\alpha, \mathsf{x})$) are constants (i.e., S is a set of constants). In order our query language to be more flexible, it will be very important to consider sorts as first class values. As a consequence, we will allow constraints of the form $\mathsf{Z}(\alpha, \mathsf{x})$, where Z is a variable ranging over sets of sorts.

References

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation. Technical report, http://www.w3.org/TR/REC-xml, October 6 2000.

[2] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the International Conference on Database Theory (ICDT'97), Delphi, Greece*, pages 1–18, Janvier 1997.

[3] S. M. Shieber. An Introduction to Unification-Based Approaches to Grammar. *volume 4 of CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University*, 1986.

[4] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A Feature-Based Constraint System for Logic Programming with Entailment. *Theoretical Computer Science*, 122:263–283, 1994.

[5] Gert Smolka and Ralf Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, April 1994.

[6] Markus Höhfeld and Gert Smolka. Definite Relations over Constraint Languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988.

[7] Hans-Jürgen Bürckert. A Resolution Principle for Constrained Logics. *Artificial Intelligence*, 66:235–271, 1994.

[8] Rolf Backofen. Regular Path Expressions in Feature Logic. *Journal of Symbolic Computation*, 17:421–455, 1994.

[9] Martin Müller, Joachim Niehren, and Andreas Podelski. Ordering Constraints over Feature Trees. In Gert Smolka, editor, *Principles and Practice of Constraint Programming - CP97, Third International Conference (CP'97), Linz, Austria*, LNCS 1330, pages 297–311. Springer Verlag, 1997.

[10] Bret Bos. The XML Data Model. http://www.w3.org/XML/Datamodel.html, 1999.

[11] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on The Web, From Relations to Semistructured Data and XML*. Morgan Kaufmann, San Francisco, California, 2000.