Safe Web Queries

(Extended Abstract)

Hong-Cheu Liu[1] and Weifa Liang[2]

[1] School of Information Systems, Queensland University of Technology,
Brisbane, QLD 4001, Australia
h2.liu@qut.edu.au

[2] Department of Computer Science, The Australian National University,
Canberra, ACT 0200, Australia
wliang@cs.anu.edu.au

Abstract. In this paper we explore the fundamental aspects of queries and computation on the Web. We first revisit several well-known declarative query languages: $CALC^{cv}$, $Datalog^{cv}$, and $Datalog^{cv}$ with negation and characterize them with respect to (eventual) computability. We then investigate the safety issue in query languages in the context of the Web and develop an algorithm for determining safe Web queries that ensure eventual computability. We finally show how to identify the fragments in each language implementable by the Web machine.

1 Introduction

The World Wide Web, temptingly targeted as a global database, is the most popular and powerful networked information system to date, which comprises vast collections of linked Web pages distributed across the Internet. The connection of the Web and databases bring many challenges and new opportunities for creating advanced database applications. Web sites are also increasingly powered by the access to their databases directly. The storage of Web information in a database can either replace or complement file storage. However, the limited access capability and loosely structured information make querying the Web significantly different from querying a conventional database [1, 2].

The theory of database queries has grown into a rich research area, which includes the expressiveness and complexity of query languages, the domain independence and the safety of queries, query translation and optimization. However, the development of Internet technology has occurred very rapidly and much of the traditional framework of database theory needs to be reinvented in the Web scenario [3]. This paper will focus on studying the theoretical foundation of Web query formalism.

The study of query languages, providing the access to the stored data, is one of the important topics in databases as well as the most deeply studied in database theory [4]. In the relational framework, three query paradigms have been developed, which are *Calculus*, *Algebra*, *Datalog.* It is well known that in classical databases some calculus queries cannot be answered sensibly, here *a query is answered sensibly* means that the values of any correct answer lie within the active domain of the query or the input [5]. To deal with this issue, researchers have developed notions of "domain independence" and "safety" to capture intuitive properties related to this finite model criterion. In the Web scenario, much of this foundation needs to be re-investigated. Our query model is along the lines of the Web query computation model of Abiteboul and Vianu [2], which views the Web as an infinite, semi-structured set of objects. Under the *infinite* nodes assumption, the expected result of a Web query could be infinite if the query is eventually computable. Due to the different scale of volume and size of the Web databases, we consider the issues of domain independence and safety in the context of the Web.

Although the design of practical Web query languages has made good progress, the study of the foundations of Web queries and languages is still in its infancy [3]. Unlike relational calculus which is the base of "relational completeness", there are no well-accepted yardsticks for expressiveness of query languages in the context of the Web.

The query languages for the Web attracted much attention in recent years. Some languages target at querying the Web as a whole, while others aim at semi-structured data and languages are targeted at querying the Web as a whole, while others are aimed at semi-structured data and XML [6–11]. We believe that a query language paradigm for the Web should support complex values due to the graph structure nature of the Web. Moreover, parameterized links are also better described in languages using an nested value approach. Based on the above arguments, in this paper we consider complex value Web query languages.

It is highly desirable to check whether or not a given query formula satisfies the "domain independence" when a new query language for any database system is proposed [12]. In this paper, we focus on *safe* queries expressed in a higher order logic applied to Web databases. The main contributions of this paper are as follows.

- We generalize the notions of "domain independence" and "safety" to Web databases and show that all safe calculus queries are domain independent and eventually computable.
- We develop an algorithm for determining whether or not a calculus query is safe. This algorithm considers both issues: domain independence and eventual computability.
- We define the notion of "continuous" for Web queries, and show that any eventually computable query is continuous. This result will be useful in checking query computability.
- We investigate the issue of whether queries expressible in the proposed complex value query languages are eventually computable. We also identify fragments of each language that can be implementable by a browser machine.

The remainder of this paper is organized as follows. Section 2 contains the backgrounds and motivations. Section 3 introduces query safety notation and develops an algorithm for determining how variables are range restricted. The notion of “continuous” is also defined, and a connection among continuity, domain independence, and (eventual) computability is explored. Section 4 investigates the computability issue of complex value query languages for the Web. Section 5 concludes the discussions.

2 Preliminaries

We use the complex value data model as a model of the Web due to it captures the Web's graph nature and the semistructured information it holds. Intuitively, the global Web database can be viewed as an infinite *complex value* data structure over the fixed schema $\mathcal{S}$ which contains *DB*, *Node* and *Node-Link* schemas.

DB is a finite set of base relations containing the relevant data such as bookmarks, local files, etc. It also contains semantic predicates that a Web query may apply. Relation *Node* = (*id, title, subject, date, length*, ...) specifies the set of Web objects. The Web objects refer to Web pages, Web sites, or other objects on the Web. Complex value relation *Node-Link* = (*source*, *Ref(label, destination, parameters)*) specifies, for each node, a finite set of links to other nodes and the labels associated with them. Sometimes, the Web allows a user to provide some parameters and returns the documents as requested. To express such a function, a *Form* will be used, which can be viewed as a document with parameters on its outgoing links.

We assume an infinite set **dom** of data values and two other sets, the set **label** of labels and the set **oid** of object identifiers. The attributes *id*, *source* and *destination* are of sorts **oid**, and attribute label of sort **label**. In the context of Web databases, the object identifiers (**oid**) will be the uniform resource locator's (URL's). Other attributes range over the underlying domain **dom**. All the results in this paper hold for multi-sorted Web databases. For simplicity, we shall restrict ourselves in the sequel to only one base sort **d**. To accommodate the values of different sorts, we may assume $\mathbf{d} = \mathbf{dom} \cup \mathbf{label} \cup \mathbf{oid}$. A Web instance $\mathbf{I}$= (**d**; *DB, Node, Node-Link*) is an infinite structure over the above schema $\mathcal{S}$.

In classical databases, it is usually required that the query be *generic*, *domain independent* and *computable*. In this paper, we investigate whether the above requirements for a query are also suitable for the Web database with complex value query language paradigm. The crucial difference between querying a conventional database and the Web is the browsing nature in the latter, which allows for navigational exploration of the Web.

A Web query q over a schema $\mathcal{S}$ is a generic mapping which associates to each Web instance $(o, \mathbf{I})$, where o is a particular object in $\mathbf{I}$, a subset of the Web node structure. The object o is the starting point (or source) of the query.

Definition 1. *A Web query q is generic iff for any Web instance $(o, \boldsymbol{I})$ and any isomorphism ρ, $q(\rho(o), \rho(I)) = \rho(q(o, \boldsymbol{I}))$.*

The above definition captures the data independence principle in databases. It means that the result only depends on the information in $\mathbf{I}$ and is independent of any particular encoding scheme.

Definition 2. *A Web query q is domain independent iff any Web instance $\mathbf{I}$ = ($\mathbf{d}$; DB, Node, Node-Link) and each pair $\mathbf{d}$, $\mathbf{d}'$, q($\mathbf{d}$; DB, Node, Node-Link) =q($\mathbf{d}'$; DB, Node, Node-Link).*

Various machines specially tailored for computations on the Web have been introduced by Abiteboul and Vianu, in particular the *browser machines* can be viewed as a Turing machine which navigates the Web using links [2]. A notion of computability for Web queries is also proposed by them [2]. We adopt this notion to our model and investigate some properties of complex value query languages in the context of the Web.

Definition 3. *[2] A Web query is finitely computable if there exists a Web machine that computes on input tape enc($\mathbf{I}$), halts and produces enc(q($\mathbf{I}$)) on the output tape.*

Definition 4. *[2] A query q is eventually computable if there exists a Web machine whose computation on input enc($\mathbf{I}$) has the following properties:*

- *The content of the output tape at each computation step forms a prefix of enc(q($\mathbf{I}$)), and*
- *For each tuple in enc(q($\mathbf{I}$)), its encoding occurs on the output tape after a finite number of computation steps.*

We now define the semantics for the complex value calculus query in the standard manner. A formula $\varphi(\boldsymbol{x})$ over $\mathcal{S}$ with an infinite domain $\mathbf{d}$ defines a Web query q which is expressed as follows.

$$q(o, \mathbf{I}) = \{v(\boldsymbol{x}) \mid \mathbf{I} \models_{\mathbf{d}} \varphi[v],\ v \text{ is a valuation over free variables of } \varphi\}.$$

3 Safe Web Queries

We consider complex value calculus which is denoted by $CALC^{cv}$. The syntax of calculus queries is the language of complex value calculus formulas. Calculus variables may denote sets. For each sort, we assume the existence of a countably infinite set of variables of that sort. A term is an atomic element, a variable, or an expression $x.A$ where x is a tuple variable and A is an attribute of x. Each (complex value) calculus formula is able to be evaluated as either true or false with respect to the input Web database and an assignment to the free variables. A complex value calculus query is a mapping from a Web database to a Web node structure expressed by an unnamed complex value relation.

We define a Web query as a (complex value) relational generic mapping that can be computed by a Web machine. We propose a particular query language, the Web calculus, which is the complex value calculus augmented with arithmetic constraints essentially. As in the first-order logic in relational setting, the Web calculus can express queries that are not computable. This raises the issue akin to *safety* in classical databases. We deal with this issue by presenting the syntactic restrictions needed to express only (eventually) computable queries. In particular, we define safe Web calculus formulas that guarantee the calculus queries are eventually computable.

3.1 Safe Calculus Queries

We now aim to develop a syntactic condition, called range-restricted, that ensures calculus queries are eventually computable, given an input Web instance. Intuitively, if a formula is *safe range*, then each variable is range-restricted, i.e., the set values assigned to each variable have a finite proof. We now define the notions of safe formula and safe term. First we define safe term.

- $R(x_1, ..., x_n)$ is a safe term if R is an n-ary relation symbol and $x_1, ..., x_n$ are variables or constants.
- If t' is a safe term, then $t = t'$, $t \in t'$, and $t \subseteq t'$ are safe terms.

Safe formula is defined as follows.

- If $R(x_1, ..., x_n)$ is a safe term, then it is a safe formula.
- If ϕ_1 and ϕ_2 are safe formulas, then $\phi_1 \wedge \phi_2$ is a safe formula.
- If ϕ is a safe formula, t and t' are safe terms, and $free(t') \subseteq free(\phi)$, then $t\ pred\ t' \wedge \phi$ is a safe formula, where *pred* is a predicate in $\{=, \in, \subseteq\}$.
- If ϕ_1 is a safe formula and ϕ_2 is finitely computable, then $\phi_1 \wedge \phi_2$ is a safe formula.
- If ϕ is a safe formula and x is a variable, then $\exists x(\phi)$ is a safe formula.

As in complex value databases, we define the set of safe-range variables of a formula using the following procedure, which returns either the symbol $\perp$ (which indicates that some quantified variable is not eventually computable) or the set of free variables that are range-restricted (i.e., each computation value of these variables has a finite proof; each variable may have finite or infinite set of computation values).

In the following, if several rules are applicable, the one returning the largest set of safe-range variables is chosen.

```
procedure safe-range($sr$)

Input: a constraint calculus formula $\varphi$
Output: a subset of the free variables of $\varphi$ or $\perp$

begin
($pred$ is a predicate in $\{=, \in, \subseteq\}$)
if for some parameterized query $\{x \mid \psi\}$ occurring as a term in $\varphi$, $x \notin sr(\psi)$
then return $\perp$
    case $\varphi$ of

$R(t)$                 : $sr(\varphi) = free(t)$
$t\ pred\ t' \wedge \psi$    : if $\psi$ is safe and $free(t') \subseteq free(\psi)$
                         then $sr(\varphi) = free(t) \cup free(\psi)$
$t\ pred\ t'$            : if $free(t') = sr(t')$
                         then $sr(\varphi) = free(t') \cup free(t)$
                         else $sr(\varphi) = \emptyset$
```

```
$\psi_1 \wedge \psi_2$              : $sr(\varphi) = sr(\psi_1) \cup sr(\psi_2)$
$\psi_1(u) \wedge \neg\psi_2(v)$ : **if** $\psi_1$ is safe and $\psi_2(v)$ is finitely computable, $v \subseteq u$
                                   **then** $sr(\varphi) = free(\psi_1)$ **else return** $\emptyset$
$\exists x \psi_1$                  : **if** $x \in sr(\psi_1)$
                                   **then** $sr(\psi) = sr(\psi_1) - \{x\}$ **else return** $\perp$
$\neg\psi$                          : $sr(\varphi) = \emptyset$
**end**
```

Note that the conjunction of the two formulas $\varphi_1 \vee \varphi_2$ is replaced by the equivalent $\neg\varphi_1 \wedge \neg\varphi_2$ before the above algorithm is applied.

We say that a formula is *safe* if $sr(\varphi) = free(\varphi)$, and a query is safe if its associated formula is safe.

Theorem 1. *Each safe-range query is domain independent.*

Proof. Sketch. According to syntactic restrictions described in the above procedure, it is easy to check domain independence criterion for each case. The theorem then follows. □

Example 1. Consider a query: `Find all node objects referencing the node object o.`

This query could be expressed as the following calculus formula.

$$\{t \mid \exists x(\mathit{Node\text{-}Link}(x) \wedge o \in x.Ref.destination \wedge t = x.source)\}$$

By the safe-range procedure, x is range-restricted due to the term *Node-Link*(x). So $x.Ref.destination$ and $x.source$ are safe terms. t is range-restricted as $t = x.source$. Therefore the above formula is safe range. This query is eventually computable.

Example 2. Query: `Find all objects which are not referenced by any other object.`

We can express this query as follows.

$$\{t \mid Node(t) \wedge \neg\exists x(\mathit{Node\text{-}Link}(x) \wedge t \in x.Ref.destination)\}$$

It is obvious that the formula is not safe by applying *sr*-procedure. So it cannot guarantee finite computability or eventual computability. Indeed the above query is *not* eventually computable.

To define the notion of "continuous" for Web queries, we first define an "approximation" of a database instance. Suppose that $\mathbf{I}_1 = (\mathbf{d}_1; DB_1, Node_1, \mathit{Node\text{-}Link}_1)$, $\mathbf{I}_2 = (\mathbf{d}_2; DB_2, Node_2, \mathit{Node\text{-}Link}_2)$, we say that $\mathbf{I}_1$ approximates $\mathbf{I}_2$, written $\mathbf{I}_1 \sqsubseteq \mathbf{I}_2$, iff $\mathbf{d}_1 \subseteq \mathbf{d}_2$, $DB_1 \subseteq DB_2$, $Node_1 \subseteq Node_2$, $\mathit{Node\text{-}Link}_1 \subseteq \mathit{Node\text{-}Link}_2$. We then define the least upper bound of a directed family of Web databases as follows. Let $\mathcal{I}$ be a set of Web database instances. We say that $\mathcal{I}$ is *directed* iff it is nonempty and for any $\mathbf{I}_1$, $\mathbf{I}_2$, there exists $\mathbf{I} \in \mathcal{I}$ such that $\mathbf{I}_1 \sqsubseteq \mathbf{I}$ and $\mathbf{I}_2 \sqsubseteq \mathbf{I}$. For a directed family of database instances $\mathcal{I} = \{\mathbf{I}_1, \mathbf{I}_2, ...\}$, $\mathbf{I}_i = (\mathbf{d}_i; DB_i, Node_i, \mathit{Node\text{-}Link}_i)$, the *least upper bound* of a collection of Web databases $\bigsqcup \mathcal{I} \overset{\text{def}}{=} (\mathbf{d}; DB, Node, \mathit{Node\text{-}Link})$, where $\mathbf{d}$ is the union of the domains of all Web databases in $\mathcal{I}$, $DB = \bigcup_i DB_i$, $Node = \bigcup_i Node_i$, $\mathit{Node\text{-}Link} = \bigcup_i \mathit{Node\text{-}Link}_i$.

Definition 5. *A query q is called continuous iff for any directed set of Web database instances $\mathcal{I}$, $\bigsqcup\{q(\boldsymbol{I}) \mid \boldsymbol{I} \in \mathcal{I}\}$ exists and $q(\bigsqcup\mathcal{I}) = \bigsqcup\{q(\boldsymbol{I}) \mid \boldsymbol{I} \in \mathcal{I}\}$.*

Here $\bigsqcup\{q(\mathbf{I}) \mid \mathbf{I} \in \mathcal{I}\}$ denotes the least upper bound of the results $q(\mathbf{I})$ with $\mathbf{I} \in \mathcal{I}$.

Theorem 2. *If a query q is eventually computable, then it is continuous.*

Proof. Sketch. Suppose it is not continuous, then $q(\bigsqcup\mathcal{I}) \neq \bigsqcup\{q(\mathbf{I}) \mid \mathbf{I} \in \mathcal{I}\}$. There are two cases: (1) there exists one tuple of $q(\mathbf{I})$ not in $q(\bigsqcup\mathcal{I})$; (2) there exists one tuple of $q(\bigsqcup\mathcal{I})$ not in $q(\mathbf{I})$, for all $\mathbf{I}$. For case (2), as $\mathbf{I}_1, \mathbf{I}_2, ...$ contain an enumeration of the nodes in the $Node$ attribute, we cannot expect to return such tuple eventually by the Web machine. For case (1), it is obvious that the query does not satisfy the monotonicity properties. This implies it is not eventually computable by the Proposition 2.3 of [2]. □

To determine whether a query is not eventually computable, it is sufficient to show whether it is not continuous.

Example 3. Consider again the query illustrated in Example 2. Let $\mathcal{I} = \{\mathbf{I}_1, \mathbf{I}_2\}$. $\mathbf{I}_1 = \{\mathbf{d}, \textit{Node} = \{A, B, C\}, \textit{Node-Link} = \emptyset\}$; $\mathbf{I}_2 = \{\mathbf{d}, \textit{Node} = \{B, C\}, \textit{Node-Link} = \{B, \{\textit{label-1}, C\}\}\}$. Then, $q(\bigsqcup\mathcal{I}) = \{A, B\}$. $\bigsqcup\{q(\mathbf{I}) \mid \mathbf{I} \in \mathcal{I}\} = q(\mathbf{I}_1) \cup q(\mathbf{I}_2) = \{A, B, C\} \cup \{B\} = \{A, B, C\}$. So $q(\bigsqcup\mathcal{I}) \neq \bigsqcup\{q(\mathbf{I}) \mid \mathbf{I} \in \mathcal{I}\}$. By Theorem 2, the query is not continuous, which implies that it is not eventually computable.

4 Computability of Web Query Languages

As mentioned in the Introduction, it is appropriate to use complex value declarative query languages in the context of the Web. We here examine some properties of the query languages. Specifically, we consider the languages $CALC^{cv}$ and $Datalog^{cv}$.

$CALC^{cv}$ queries are formulas with a set of variables of specified sorts. The result of a $CALC^{cv}$ query is a set of tuples of complex value data which specifies the return node structure.

Query languages based on the deduction paradigm are extensions of Datalog in order to incorporate complex values into them. Those languages based on the calculus do not increase the expressive power of the ALG^{cv} or CALC^{cv} [4]. We denote Datalog for complex values and Datalog for complex values with negation as $Datalog^{cv}$ and $Datalog^{cv,\neg}$ respectively. However, some queries can be expressed in this deduction paradigm more efficiently and with lower complexity than they can be by using the power-set operator in the ALG^{cv}.

Definition 6. *A database clause (rule) is an expression of the form*

$$p(t) \leftarrow L_1, ..., L_n,$$

where the head p is a derived predicate, and each L_i in the body is a literal. A program $\mathcal{P}$ is a finite set of rules that are extended with functions.

We distinguish between the *intentional* predicates and functions, which appear in the head of a rule, and the *extensional* one, which appears only in the body.

Definition 7. *A Datalog query is a pair* $(Q, \mathcal{P})$ *where* Q *is a formula of the form* $\leftarrow W$, *where* $W \in CALC^{cv}$, *and* $\mathcal{P}$ *is a finite set of rules that includes definitions of predicate symbols which appear in* Q.

Definition 8. *A rule is safe if each variable appearing in the head appears in the body, and the body is a safe formula.*

Theorem 3. *All safe-range* $CALC^{cv}$ *and safe-range* $Datalog^{cv,\neg}$ *queries are eventually computable.*

Proof. Sketch. Since $CALC^{cv}$ is equivalent to $Datalog^{cv,\neg}$ with stratified semantics, we only consider $Datalog^{cv,\neg}$ here. According to our safe definition, a query defined by safe stratified $Datalog^{cv,\neg}$ can be eventually computed by a Web machine by reading the input and evaluating the program. In particular, in the case of negation occurring in a program rule, $\psi_1 \wedge \neg\psi_2$ is eventually computable as long as ψ_2 is finitely computable. □

As *browsing* is a prevalent style on the Web, we consider fragments of $Datalog^{cv,\neg}$ and $CALC^{cv}$ that can be computed by a browser machine. We provide a syntactic restriction on variables, which limits the range of objects reachable from the source node.

Definition 9. *The set of source-range-restricted variables in a Datalog rule* r *is a subset of variables in* r *satisfying*

- *For each tuple of relation Node-Link, if* x *referring to the attribute 'source' is source-range-restricted, then variables referring to the attribute 'destination' are also source-range-restricted.*
- *If* R *is some intentional database predicate and* x *is one of the variables of* R *referring to the attribute with sort* **oid**, *then* x *is source-range-restricted.*

A rule is *source-safe-range* if all its variables are source-range-restricted. A program is *source-safe-range* if all its rules are both safe and source-safe-range.

Theorem 4. *All source-safe-range* $Datalog^{cv,\neg}$ *and* $CALC^{cv}$ *queries are eventually computable by a browser machine.*

Proof. Sketch. Let P be a source-safe-range $Datalog^{cv}$ program. By source-safe-range syntactic restriction on P, the evaluation of P only needs node objects reachable from the source. As in Datalog, a simple induction will apply to the evaluation. This ensures that the query defined by P be eventually computable by a browser machine by using the load/compute technique described in [2]. □

Example 4. The following Datalog program is source-safe-range.

$$
\begin{aligned}
&answer(source) \leftarrow \\
&\quad answer(t) \leftarrow answer(y) \wedge Node\text{-}Link(x) \wedge y = x.source \\
&\qquad\qquad \wedge t \in x.Ref.destination
\end{aligned}
$$

The above query asks for all objects reachable from the source. Since x and y are safe range, t is also safe range. y is source-range-restricted. t is also source-range-restricted because t refers to the *destination* attribute. The above program is safe. Therefore the program is source-safe-range. By Theorem 4, the query is eventually computable by a browser machine.

Theorem 5. *Every query in source-safe-range $Datalog^{cv,\neg}$ with inflationary semantics is eventually computable by a browser machine.*

Proof. Similar to the proof for source-safe-range $Datalog^{cv,\neg}$. □

5 Conclusion

In this paper we presented a new complex value model of query and computation on the Web, closely related to, but significantly different from previous works. Accordingly, we considered complex value query languages including $CALC^{cv}$ and $Datalog^{cv,\neg}$.

We adopted a general approach to restricting the Web calculus language with negation to a safe subset that contains only formulas that are eventually evaluable on any Web database input. However, there exist some eventually computable queries which are not safe. In classical databases, any definition of safety must leave out some queries that are evaluable. Unfortunately, this phenomenon also occurs in the context of the Web. Nevertheless, we identified the safe complex value queries that greatly extend the expressive power of first order queries.

We addressed a connection between continuity and eventual computability which provides more in depth understanding and is useful to the characterization of queries with respect to computability.

The results of this paper could be applied to the design of more practical languages. In particular, we believe that the complex value should be supported in Web query languages, and the properties of safe Web queries explored in this paper provide guides to extend existing languages.

Finally, the issues of optimization techniques, expressive power and complexity analysis for Web queries need to be investigated. The Web raises vast opportunities and challenges for database applications. Much of the traditional framework of database theory needs to be revised or reaffirmed. This paper tackled the issue of safe queries with respect to computability.

References

1. Mendelzon, A., Milo, T.: Formal models of web queries. Information Systems **23** (1998) 615–637
2. Abiteboul, S., Vianu, V.: Queries and computation on the web. Theoretical Computer Science **239** (2000) 231–255
3. Vianu, V.: A web odyssey: from codd to **XML**. In: Proceedings of ACM Symposium on Principles of Database Systems. (2001) 1–15

4. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
5. Topor, R.: Domain independent formulas and databases. Theoretical Computer Science **52** (1987) 281–307
6. Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization techniques for unstructured data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. (1996) 505–516
7. Mihaila, G., Mendelzon, A., Milo, T.: Querying the world wide web. Journal of Digital Libraries **1** (1997) 68–88
8. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.: The lorel query language for semi-structured data. Journal of Digital Libraries **1** (1997)
9. Deutsch, A., Fernandez, M., Florescuall, D., Levy, A., Suciu, D.: A query language for **XML**. In: Proceedings of WWW8. (1999) 11–16
10. Chamberlin, D., Robie, J., Florescu, D.: Quilt: An **XML** query language for heterogeneous data sources. In: Proceedings of WebDB. (2000) 53–62
11. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web. Morgan Kaufmann Publishers (2000)
12. Liu, H.C., Yu, J.: Safe database queries with external functions. In: Proceedings of International Database Engineering and Applications Symposium, Montreal, Canada (1999) 260–269