Formal Models of Web Queries *

Extended abstract

Alberto O. Mendelzon
University of Toronto
mendel@db.toronto.edu
http://www.cs.toronto.edu/~mendel

Tova Milo
Tel Aviv University
milo@math.tau.ac.il
http://www.math.tau.ac.il/~milo

Abstract

The paper presents a new model of query and computation on the Web. We focus on two important aspects that distinguish the access to Web data from the access to a standard database system: the navigational nature of the access and the lack of concurrency control. We show that these two issues have significant effect on the computability of queries. To illustrate the ideas and how they can be used in practice for designing appropriate Web query languages, we consider a particular query language, the *Web calculus,* an abstraction and extension of the practical Web query language *WebSQL.*

1 Introduction

Tools and techniques for retrieving information from the World Wide Web are rapidly being developed [14, 15, 16, 5]. Most of these works are based on the metaphor of the Web as a database, in order to carry over and adapt familiar query languages such as SQL or Datalog. However, the Web is *not* a database, and querying it differs significantly from querying a conventional database. Two essential aspects that make a query on the Web different from a query on a standard database are the lack of concurrency control and the limited data access capabilities.

First, consider the impact of the lack of concurrency control on which queries can and cannot be effectively evaluated. Concurrency control on the Web is very crude: the owner of a document can lock it to prevent others from accessing it while it is being modified, but no other user can do this, and there is no transaction mechanism. This makes many queries in principle impossible to compute in a finite time. For example, finding the documents reachable directly or indirectly from a given starting point may require following a chain in which documents are being added to the end of the chain faster than we can discover them, so the computation may never terminate.

*Supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario. The work of the second author was supported by the Israel Academy of Sciences and Humanities.

To appear in the Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems, Tucson, Arizona, May 1997.

Since the dynamic nature of the Web has such an effect on query computability, it is natural to ask what happens if we assume a static Web. For example, stronger concurrency control mechanisms in future versions of the Web might isolate a query from changes and provide a conceptually static view. What will be the effect of introducing such a mechanism on query evaluation? Even when updates to the Web can be ignored, many queries that would easily be computable on a conventional database are not so on the Web, because of the limitations in data access: access to Web documents is strictly navigational. There are only two ways to retrieve a document; either through its URL (uniform resource locator), or through some other document (possibly a form) that points to it.

This statement may seem false at first sight to anyone who has used an *index server* such as *Altavista* [12]. These are Web sites that provide associative access to a large collection of documents by using a searchable index (built offline by navigation). However, from a theoretical point of view, these indices do not provide true associative retrieval from the whole Web – they do not guarantee they have indexed every document, nor do they give any guarantee of currency of the index. They are basically forms that provide access to some pre-computed data.

This limitation in data access has a strong impact on the feasibility of queries: a query such as "are there any documents pointing to document d?" is impossible to evaluate because, even in a static Web, there is no way to examine *all* documents and be sure that we have not missed any.

In this paper we propose a formal model of Web queries that highlights the effects that the limited access to data and the lack of concurrency control have on data retrieval. Our goal is to provide a notion of computable Web queries that will play the role that the computable queries of Chandra and Harel [9] have played in the theory of relational databases. Of course, there are other important aspects of the Web, such as heterogeneity, autonomy, and lack of structure, which we do not address here. Nevertheless, we believe that the model we propose can serve as a robust basis for the theoretical study of these and other issues.

To isolate the problems caused by the limited access to Web data from those caused by the lack of concurrency control, we start by analyzing the static case, that is, we ignore updates. We model the Web as a relational database containing a relation of *Node* objects, one per document, and a relation of *Link* objects, one per node-to-node link, plus possibly additional relations called *base relations*, containing auxiliary information, such as sets of initial URL's where the navigation may start.

In Section 3 we present our notion of computable queries on the Web, which we call *Web queries*, as relational queries that can be computed by a special kind of Turing machine, a *Web machine*, that can only access the Web (i.e. the Node and Link relations) through an oracle that provides the navigation functions. We give two characterizations of Web queries, one in terms of the sub-Web they depend on, and one in terms of generic machines[1]. In Section 4, we consider *eventually computable* queries, that can be evaluated only by a non-terminating computation. We characterize them by a monotonicity property and by a new kind of generic machine.

To illustrate the ideas and how they can be used in practice for designing appropriate Web query languages, we consider in Section 5 a particular query language, the *Web calculus,* an abstraction and extension of the practical Web query language *WebSQL* [16]. We show that, if not properly restricted, the language can express queries that are not Web queries (e.g. "find all nodes with no incoming links"), and present the syntactic restrictions needed to express only Web queries, and various classes of eventually computable queries.

Once the effect of the limited access to data is understood, we move in Section 6 to the issue of updates and lack of concurrency control. To account for the dynamic Web environment we now view a Web as an infinite sequence of *Node* and *Link* relation pairs, instead of a single pair. A *dynamic Web machine* is exactly like a Web machine, except that the oracle can, each time it is invoked, nondeterministically switch from one *Node, Link* pair to the next one in the sequence. We distinguish three classes of queries in the dynamic Web environment, and analyze their relationship to their static-model counterparts.

Finally, in Section 7 we examine the effect of Web *forms* on these results, both in the static and dynamic cases. We show that, with appropriate assumptions, the results can be extended to handle forms and index servers. We conclude in Section 8. For lack of space we omit proofs.

Related work

The only work we are aware of dealing with formal models of Web queries is [2], which is closely related to ours. There is however one central difference. According to [2], "Perhaps the most fundamental aspect of our model is that we view the Web as infinite." The argument for this is that "exhaustive exploration is –or will soon become– prohibitively expensive." This is not clear to us. First, the Web is *finite* at any given moment, even if large. Thus we believe it should be modeled as such. Moreover, the Altavista index server has indexed, as of October 1996, about 30 million documents stored on 225,000 servers. The index is about 30 Gigabytes. By database standards, this is not very large.[1]

In our model, the Web is finite, but there is still non-terminating behavior, caused by two very different reasons treated separately: the restriction to navigational access, and the lack of concurrency control. This leads to a different classification of queries from the one in [2], highlighting the specific effects of these two issues. For example, the query "find all documents reachable from document d" is computable in our static model, but not in the model of [2]. In the dynamic Web, this query is only eventually computable, as it is in [2]. On the other hand, a query such as "list all documents" is eventually computable in both their model and ours (static and dynamic). And a query such as "does this node have any incoming link?" is not computable at all in any of the models.

A further difference is that our analysis, by allowing parameterized links, captures forms, which they do not treat. Also, our way to capture index servers is more accurate because it does not assume they are "perfect" indexes.

Other relevant work includes query languages for graph databases and hypertext documents[6, 11, 17, 18] as well as query languages for structured or semi-structured documents such as [13, 4, 10], and for unstructured data[8].

2 Preliminaries

We use a simple relational model of the Web to best highlight the essential computational issues. We first describe informally the main ideas, and then present our formal data model.

Documents and Links: The World Wide Web is a large, heterogeneous, distributed collection of documents connected by hypertext links. At the highest level of abstraction, it can be viewed as a graph whose nodes are Web objects that are identified by a Uniform Resource Locator (URL) and have some arbitrary content whose interpretation depends on its type (HTML, Postscript, image, audio, etc.). Also, Web servers provide some additional information such as the type, length, and the last modification date of an object. Moreover, an HTML document has a title and a text. So, for query purposes, we can associate a Web object with a tuple in a virtual Nodes relation:

$$N[id, title, content, type, length, modif, ...]$$

The *id* represents the URL and is thus a key, and all other attributes may be null.

HTML documents may contain hypertext links to (some offset within) other documents. We capture the information present in a link as a tuple in a Links relation:

$$L[source, destination, d\_offset, \ldots]$$

where *source* and *destination* are the Oid's of the origin and destination of the link. Additional attributes can describe link properties such as label. [2]

Forms: Forms are special Web documents allowing a user to fill in some parameters and get as a result another document. A form can therefore be viewed as a document whose outgoing links have parameters. The (virtual) documents pointed to by the parameterized links are the ones returned by the program associated with the form when called with the given arguments. Such parameterized links can be described in a relation L_p having the same structure as the links relation L with an additional attribute *param_set*. Taking a *nested values* approach, *param_set* could contain a set of tuples, one for each parameter name and values. To stay within the relational model, we can model this by having *param_set* contain some id for the parameter set, and using an auxiliary parameters relation

$$P[param\_set : t_1, param\_name : t_2, param\_val : t_3]$$

[1] For example, the Big Sur project at Berkeley [7], dealing with data from NASA's Earth Observing System, is designed for 1 Terabyte of data, or about 25 times the size of the Altavista index, and this is only 1/1000 of the ultimate data size that the EOS project will have to handle.

[2] In fact we should also add an *offset* within the source document to allow for the possibility of the same link appearing several times in one document.

recording the relationship between the set id's and the actual parameter values. Note that index servers often have a form-based interface, thus they can also be modeled as nodes with outgoing parameterized links.

Base relations and data access: People interacting with the Web often use additional data such as bookmarks, local files, and perhaps even some data stored in a database system. This data can be used to initiate a Web access, for example by following some selected bookmark. Abstractly, one can view all this additional information as a set of *base relations* containing the relevant data. The main difference between these relations and the N, L, L_p, and P relations is that the only way to access the Web relations is by explicitly specifying a URL, or following a link; while the base relations are fully accessible, say, using a relational query language. Note that, although index servers give the illusion of content search capabilities, they only provide an approximation to it. When an index server is queried, the query is typically evaluated on a pre-computed index, and the answer does not necessarily reflect the state of the Web at query time, or a complete and consistent state of the Web at any time.

Finiteness: The Web is very large, but at any given time, the set of documents in it, and the amount of data in them, are finite (although they keep changing and growing with time). Ordinary (non-form) HTML documents contain a finite number of outgoing links. For forms, this is not obvious, since they may have parameterized links. However, in practice, the number of possible combinations of parameter values that actually lead to documents is usually finite. (In particular, most parameter fields have a fixed size and thus can be fed only a bounded, but possibly very large, number of values.) We will thus assume in the sequel that the Node, Link, Parameterized-link, and Parameters relations N, L, L_p, P are finite at any given point in time. We shall deal with the changes over time to these relations separately. Some forms may accept input of arbitrary size and may generate infinitely many documents (e.g. a form that is an HTML editor). We believe however that such forms will rarely be used when querying the Web, and thus we mostly ignore them in this work. Nevertheless, we will point out in the sequel whenever the finiteness assumption for forms makes a difference.

We are now ready to present the formal data model. To simplify, we ignore for the moment the issue of forms and parameterized links and present a simplified version of the model. We shall come back to this issue in Section 7.

We assume an infinite set D of *data values*, and a finite set T of *simple types* whose domains are subsets of D. Tuple types $[a1 : t1, ..., a_n : t_n]$ with attributes a_i of simple type t_i, $i = 1 \ldots n$ are defined in the standard way. The domain of a type t is denoted by $dom(t)$. We distinguish a simple type $Oid \in T$ of *object identifiers*, and two tuple types

$Node = [id : Oid, ..., a_i : t_i, ...]$

and

$Link = [source{:}Oid, \ldots, destination{:}Oid, \ldots, b_j{:}t_l, \ldots]$

The domain $dom(Oid)$ is assumed to be infinite and recursively enumerable, and the attribute names in the two definitions are all distinct. We assume that none of the attributes other than *id*, *source*, and *destination* are of type Oid. We shall refer to tuples of the first type as *Node objects* and to tuples of the second type as *Link objects*. In our model of the World Wide Web, documents will be mapped to *Node* objects and the hypertext links between them to *Link* objects. In this context, the object identifiers (Oid) will be the URL's.

A **Web database schema** W is a relational database schema with a finite set of base relations $DB = \{R_1, \ldots, R_n\}$, and two additional relation schemes, N containing node objects, and L containing link objects. An **instance** of a Web database schema W, also called a **Web database**, is a mapping associating a finite set of tuples with every relation name $R_i \in DB$, a **finite** set of node objects with N and a **finite** set of link objects with L, such that the following holds: (1) *id* is a key in relation N, and (2) there is a referential integrity constraint between the *source* attribute in L and the *id* attribute in N.

Note that the referential constraint is only for the *source* attribute and not for the *destination*. This is because documents can contain URL's pointing to non existing documents. When clear from the context, we shall overload notation and use $W = (DB, N, L)$ to denote both the Web database schema and the instance.

3 Queries

We now formalize the notion of queries on Web databases. The standard definition requires queries to be generic and computable by a Turing machine on some encoding of the database. While in ordinary databases the user may have full control of the database relations, and thus a standard Turing machine is an adequate abstraction of the computation power, in the Web context the access to data is limited to navigation. We introduce *Web machines* to formalize this limitation.

Definition 3.1 *A* **Web machine** *is a Turing machine augmented with an oracle. It has two input tapes, an ordinary tape on which the input DB base relations are encoded, and an oracle tape on which the N and L relations are encoded. It also has two working tapes, an ordinary working tape and an oracle working tape. Finally, it has an output tape on which the result of the computation is written in an* append only *manner.* [3]

At each point of the computation, the machine can do one of the following:

1. *read from the ordinary input tape or from any of the working tapes (it cannot read from the oracle input tape),*
2. *write on the output or any of the working tapes,*
3. *call the oracle.*

In the last case, the oracle reads the content of its working tape, interprets its content as a node id, and copies (from its input tape) to the working tape the relevant (encoded) node tuple, if a node with such an id exists, and also the tuples of all its outgoing links. If no such node exists, the oracle simply writes a special $\perp$ symbol on the tape.

The machine captures the idea that nodes can be accessed only in a limited way – by explicitly specifying their id. To navigate starting from a given URL, the oracle is first called for this URL. It returns, among other things, the id's of documents pointed to by this document. Then it can be called again on these id's to access their data, and so on.

[3] The use of multiple tapes and the *append only* manner of the output tape is insignificant. It is used only to simplify the presentation. A similar definition can be given for a single ordinary tape.

Definition 3.2 *A* **total relational query** *is a generic mapping Q from instances of a database schema to instances of another one, such that there exists a Turing machine that, given any instance I of the first schema and any encoding $enc(I)$ on its input tape, the machine halts with $enc(Q(I))$ on its output tape.*

Note that this is more restrictive than the traditional definition of *relational queries* in [9], that allows partial recursive functions. For brevity, we omit the word *total* in the rest of this paper, and whenever we say relational query we mean a total one.

Definition 3.3 *A* **Web query** *is a relational query Q mapping Web database instances to sets of tuples over the values in the instances, such that there exists a Web machine that computes Q on any Web. That is, given any Web database $W = (DB, N, L)$, and any encoding enc, when started with $enc(DB)$ on its ordinary input tape and $enc(N)$, $enc(L)$ on its oracle input tape, the machine halts with output $enc(Q(W))$ on its output tape.*

The following are examples of Web queries:

- List the title of nodes reachable from the node with oid o by a path of length less than 3.
- Find all nodes on a cycle of length at most 3 containing the node with oid o.
- Find the nodes reachable from the node with oid o.

Note that the last query above is a Web query by our definition but is not computable according to [2] because they assume that the Web is infinite.

We next characterize Web queries. From the characterization (in Theorem 3.1 below) it will follow that queries like the following are not computable in the Web context (even though they are computable in the standard database context).

- Find all nodes.
- Find all nodes referencing the node with oid o.
- Find all nodes with no incoming links.

Before presenting the characterization we define an equivalence relation between Web instances. Given a set S of values, we say $W =_S W'$, for two Web instances W, W' over the same schema, if (1) W, W' agree on the contents of all the base relations, and (2) the contents of the Nodes and Links relations in W and W' are identical when restricted to the nodes with id's in S, the nodes reachable directly or indirectly from such nodes in W or W', and the outgoing links of all these nodes.

Theorem 3.1 *A relational query Q, mapping a Web database to a set of tuples over the values in the Web database, is a Web query iff for every two inputs $W = (DB, N, L)$, and $W' = (DB, N', L')$ such that $W =_S W'$, where S is the set of values mentioned in DB, it is the case that $Q(W) = Q(W')$.*

Basically this means that Web queries are only interested in the sub-Web reachable using data in the base relations plus data found along the way, and ignore all other nodes. A Web query can thus always be computed in two phases: First all the reachable documents are retrieved, and then the query is evaluated on them (and the base relations) without any further access to the Web. (Proof omitted.)

Remark: To deal with the use of constants in queries, one needs to talk about C-generic queries rather than just generic queries[9]. In the same way, we can naturally extend our definition and talk about C-generic Web queries – these are simply C-generic relational queries that are computable by a Web machine. C-genericity allows the use of constants, so in particular constant URLs can be used in queries. Once this is allowed, the above theorem becomes:

Theorem 3.2 *(Theorem 3.1 - revisited) A C-generic relational query Q, mapping a Web database to a set of tuples over the values in the Web database, is a C-generic Web query iff for every two inputs $W = (DB, N, L)$, and $W' = (DB, N', L')$ such that $W =_S W'$, where S is the set of oid's values in DB and C, it is the case that $Q(W) = Q(W')$.*

□

The following is an immediate consequence of the above theorem. We say that a Boolean Web query Q is trivial if either for every instance I $Q(I) = \emptyset$, or for every instance I $Q(I) = \{[\,]\}$.

Corollary 3.3 *When there is no input database and no constants, Web queries are trivial.*

The above theorem characterizes Web queries as a special sub-class of relational queries. Is it possible to give the Web queries a direct characterization independent of relational queries?

For standard relational queries, such characterizations exist in the literature, and are given in terms of languages [9] or machines [3] that compute exactly all the relational queries. It turns out that a similar characterization can be given for Web queries as well. For that we use *Web relational machines*, a slight refinement of Abiteboul and Vianu's *reflective relational machines* [1]. We next briefly recall the definition of such machines and then define Web relational machine.

Definition 3.4 *A* **reflective relational machine** *is a Turing machine with a special query tape and a separate* relational store*, capable of storing arbitrary relations. Initially, the relational store contains only the input relation R_0. The machine computes as an ordinary Turing machine, except that whenever the machine enters a special "query state," the contents of the query tape are interpreted as a first-order query, which is evaluated in a single step. The results of the evaluation are either stored in another relation in the relational store or, if the query was a Boolean query, communicated to the machine on the query tape.*

Definition 3.5 *A* **Web relational machine** *is a variant of the reflective relational machine. It gets a Web database as input* [4] *and works like an ordinary reflective relational machine, except that the only two queries it can use to access the relations N and L are: $\{\ n_1, ..., n_i \mid R(n_1) \wedge N(n_1, ..., n_i)\ \}$ and $\{\ l_1, ...l_j \mid R(l_1) \wedge L(l_1, ..., l_j)\ \}$, where R is some fixed relation name different from N, L.*

It was shown in [1] that reflective relational machines compute exactly all the relational queries. The following theorem shows a similar relation between Web relational machines and Web queries.

Theorem 3.4 *Web relational machines compute exactly all the Web queries.*

[4] Note that reflective machines have only one input relation; this is not an important difference, since we can always take the Cartesian product of all input relations, then use projection.

4 Eventually computable queries

There are many relational queries that are not Web queries. Examples of such queries are "list all documents referencing o," or "list the titles of all articles." The termination of computation of such queries cannot be guaranteed. Nevertheless, it is possible to effectively produce all the tuples in their output (though we may never know that we are done). For example, the query "list all documents referencing o" can be computed by enumerating all possible URL's and for each one testing whether it corresponds to an existing node, and whether this node point to o; but there is no way to stop. Following [2] we call such queries *eventually computable.*

To formally define this, we first explain what it means for a relational query Q to be *eventually computed* by a Web machine.

Definition 4.1 *Let Q be a mapping from instances of a Web database to relations. We say that Q is* **eventually computable** *by a Web machine if there is a Web machine whose computation on any Web database encoded as $enc(W)$ has the following properties:*
(1) the contents of the output tape at each step of the computation form a prefix of $enc(Q(W))$, and
(2) for each tuple in $Q(W)$, its encoding appears on the output tape after a finite number of steps in the computation.

The notion is the same as in [2], except that our Web machine is different from theirs. We now use it to define the **eventually computable** Web queries.

Definition 4.2 *An* **eventually computable** *Web query (EC Web query, for short) is a relational query (in the sense of Definition 3.2) mapping Web database instances to sets of tuples over the values in the instances, that is eventually computable by a Web machine.*

The notion is again similar to that of [2], but there is one significant difference - our definition of EC Web queries requires Q to be a relational query, not an arbitrary function. We argue that this is essential if we want to distinguish between non-computability caused by the limited access capabilities of the Web, and non-computability that has only to do with the complexity of the function being evaluated (independently of whether the input is ordinary relational data or Web data).

The two queries above, "list all documents referencing o," and "list the titles of all articles," are examples of EC Web queries. On the other hand, we shall show below that the query "Find all nodes with no incoming links" is not an EC Web query. Intuitively this is because there is no way to ensure that we have examined *all* nodes to conclude that some node has no incoming links. Formally, this will follow from Theorem 4.1 below.

As for Web queries, we give a characterization of the sub-class of the relational queries that are EC Web queries. The characterization is based on a monotonicity property: at each point of the computation, the machine has seen only a part of the Web and does not know what the rest of the Web looks like. Thus if a tuple is output at some point, it will also be output whenever the query is evaluated in a network "containing" the sub-network seen so far. This notion of containment has to be defined with some care.

Let S be a set of values. Define a class $\mathcal{W}_S$ of Web instances, and a partial order $\leq_S$ on them, as follows: (1) An instance that contains nodes for all the oid's in S (plus perhaps some more nodes), is in the class. (2) If W, W' are in the class, and W' is obtained from W by adding a set of nodes N', and a set of outgoing links L' for them, such that the nodes in N' are unreachable (in the extended instance) from nodes in W, then $W \leq_S W'$.

Theorem 4.1 *A (C-generic) relational query Q, mapping a Web database to a set of tuples over the values in the Web database, is an EC Web query iff for every input $W = (DB, N, L)$, Q is monotone with respect to $\leq_S$, where S contains all the values mentioned in DB (or C).*

The theorem implies that every EC Web query Q can be computed in a monotone manner as follows:

- First evaluate Q with respect to to the sub-Web reachable using values in DB (and C) and output the result, and then
- start enumerating oids, and whenever finding a new node (1) recompute Q with respect to the sub-Web reachable using this oid plus all values in the sub-Web of the previous step, and (2) add to the output all new tuples.

Theorem 4.1 assures that if a tuple is output in one of the iterations it will also be in the output of later iterations and, in particular, is still in the output when the whole Web has been seen. The fact that Q is a relational query means that it is computable in the traditional sense and thus each iteration step will halt.

As for Web queries, an independent characterization for EC Web queries also exists, and can be given in terms of a special variant of reflective relational machines.

Definition 4.3 *A* **monotone relational machine** *is a reflective relational machine that takes as input a Web database $W = (DB, N, L)$ and works in two phases.*

phase 1: *For all subsets S_i of values from W including all the values in DB, compute the sub-Web N_i, L_i of N, L, containing all the nodes reachable from nodes with id in S_i, and all their outgoing links.*

phase 2: *Run some reflective relational machine on all the instances $W_i = (DB, N_i, L_i)$ (the same machine on all the instances), and output the union of the results.*

Theorem 4.2 *Monotone relational machines compute exactly all the EC Web queries.*

5 A Web calculus

In this section we study a particular query language in light of the various classes of queries described in the previous sections. Our language, which we call the *Web calculus*, is an extension and an abstraction of an implemented language, *WebSQL*, described in [16]. WebSQL integrates content-based retrieval, as provided by index servers on the Web, with structure and topology-based retrieval. For example, suppose we want to find documents containing the string "database" that can be reached from the Computer Science Department's home page through paths of length 2 or less, without leaving the local server. The following query expresses this.

SELECT	$d.url, d.title$
FROM	Document d SUCH THAT
	"www.cs.toronto.edu" $= \mid\Rightarrow\mid\Rightarrow\Rightarrow d$
WHERE	$d.title$ CONTAINS "database";

The regular expression $= |\Rightarrow| \Rightarrow\Rightarrow$ restricts the path starting at the URL "www.cs.toronto.edu" to have zero, one, or two "local" links, i.e., links pointing to documents on the same server. The semantics of WebSQL is given using *Web calculus*. The Web calculus for Web database schema (DB, N, L) is the set of first-order formulas with equality on the following vocabulary:

- A predicate symbol R_i for each base relation in DB, with the same arity as the relation;
- Predicate symbols N and L, with the same arity as the corresponding relations;
- A ternary predicate $Path(n_1, R, n_2)$; the semantics is that n_1 and n_2 are nodes from the nodes relation N and there exists a path between them using links in the links relation L. Furthermore, R is a regular expression on an alphabet of link types, and the links along the path are restricted to satisfy R. We will not be making use of the regular expression in this section; see [16] for details.
- A binary *containment predicate* that takes as first argument an oid and as a second argument an atomic value; it means that the string represented by the second argument occurs within the body of the document represented by the first argument. We write it in infix notation as n `contains s`.

However, we need to restrict the syntax so that only computable or eventually-computable queries are expressible. For example, in the full language a query such as "find nodes with no incoming links" is easily expressible as

$$\{x \mid N(x, \ldots) \wedge \forall y (N(y, \ldots) \rightarrow \neg L(y, x, \ldots))\}$$

but we know this query is not eventually computable by Theorem 4.1.

In this Section, we give a syntactic restriction of the Web calculus, called the *safe* Web calculus, and show that it captures exactly the Web calculus queries that are Web-computable; we also give a weaker restriction that captures exactly the queries Q such that both Q and its complement are eventually computable; and finally, an even weaker one that guarantees that all queries that satisfy it are eventually computable.

Two kinds of restrictions are needed to weed out formulas that express non-computable Web queries. First, we have to ensure that the first argument in a N, L, or $Path$ atom is bound to a known set of nodes; this is similar to what is called *groundedness* in [16], and *source-safety* in [2]. Second, as in the usual relational calculus, we have to ensure that arguments of a negated atom are instantiated and that terms of a disjunction use the same set of variables. The restrictions are specified as follows.

Definition 5.1 *A formula in the* **safe** *Web calculus is a formula in the Web calculus which is of one of the following forms. In the list below, a is always a constant,* $x, x_1, \ldots, x_n$, $y, y_1, \ldots, y_m$ *are variables or constants, and* $\phi(x_1, \ldots x_n)$ *and* $\phi'(x_1, \ldots x_n)$ *are safe formulas.*

- $N(a, x_1, \ldots, x_n)$, $L(a, x_1, \ldots, y_n)$, $Path(a, R, x)$, $R_i(x_1, \ldots, x_n)$, $x = a$
- $(\phi(x_1, \ldots x_n) \vee \phi'(x_1, \ldots x_n))$,
$(\phi(x_1, \ldots x_n) \wedge \phi'(y1, \ldots y_m))$,
$(\phi(x_1, \ldots x_n) \wedge \neg \phi'(x_{i_1}, \ldots, x_{i_j}))$
- $(\phi(x_1, \ldots x_n) \wedge x_i = x_j)$,
$(\phi(x_1, \ldots x_n) \wedge x_i \neq x_j)$,
$(\phi(x_1, \ldots x_n) \wedge x_i = y)$,
$(\phi(x_1, \ldots x_n) \wedge x_i$ `contains` $\mathtt{x_j})$,
$(\phi(x_1, \ldots x_n) \wedge x_i$ `contains a`$)$,
$(\phi(x_1, \ldots x_n) \wedge L(x_i, y_1, \ldots y_m))$,
$(\phi(x_1, \ldots x_n) \wedge N(x_i, y_1, \ldots y_m))$,
$(\phi(x_1, \ldots x_n) \wedge Path(x_i, R, y))$
- $\exists x_i \phi(x_1, \ldots x_n)$

The next theorem shows that safe formulas capture all the Web queries expressible in the Web calculus.

Theorem 5.1 *A Web calculus formula expresses a Web query if and only if it is equivalent to a safe Web calculus formula.*

The *if* direction is straightforward. For the *only if* direction, we use Theorem 3.2. The idea is that one can construct a safe formula computing all the nodes reachable from nodes with id's in DB or constants in the query, and then use it to rewrite the original formula so that all node variables range only over this domain.

Clearly, there are EC Web queries, such as "list all the nodes in the Web," that are not expressible as safe calculus formulas. We thus define *weakly safe* formulas, that express a large set (but not all) of the EC Web queries expressible in the Web calculus. These are defined using *semi safe* formulas that capture exactly those queries expressible in the Web calculus such that both the query and its complement are EC.

Definition 5.2 *A Web calculus formula* $\phi(x_1, \ldots, x_n)$ *is* **semi-safe** *if the formula* $\phi(a_1, \ldots, a_n)$ *is safe, where the* a_i*'s are constants.*

For example, the query above, list all the nodes, can be expressed by the semi-safe formula $N(x, \ldots)$. It is not hard to see that semi-safe formulas express only eventually computable Web-queries. However, there are eventually computable Web-queries in the calculus that are not expressible in this way. For example: find all nodes with some incoming links. This is expressible as

$$N(x, \ldots) \wedge \exists y N(y) \wedge L(y, x, \ldots))$$

which is not semi-safe. The fact that this query is not expressible by a semi-safe formula follows from the next theorem.

Theorem 5.2 *A Web calculus formula* ϕ *is equivalent to a semi-safe formula if and only if both the query expressed by* ϕ *and its complement are EC Web queries.*

Finally, we define a class of queries that allows the expression of more eventually computable Web queries expressible in the calculus (but not all of them).

Definition 5.3 *A Web calculus formula is* **weakly safe** *if it contains no universal quantifiers and every negated subformula of it is semi-safe.*

For example, note the formula given above for the query "find all nodes with incoming links" is weakly safe.

It is easy to see (by induction on the structure of the weakly safe formulas) that all the queries expressed by such formulas are eventually computable. The proof that it does not express all such queries follows from a characterization theorem for weakly safe queries, which is omitted here. It is open whether there are syntactic restrictions that capture exactly all the eventually computable Web calculus queries.

Finally, we note that all the queries expressible in the WebSQL language of [16] are EC Web queries, and those satisfying the "groundedness" restriction in that paper are Web queries.

6 Dynamic environment

The Web keeps changing and growing. In the current architecture no global concurrency control mechanism exists and thus the computation of queries is affected by the dynamic environment.

One issue is that a query that accesses the same document several times may see different things because the document was updated between the accesses. To some extent this can be avoided by using local document storage – whenever a document is read it is stored on the local disk, and further access to the document can be directed to this storage. This is a common practice in many Web tools.

A computationally more significant issue is that queries that seem rather simple and computable may never terminate. For example consider the query "list all documents reachable from o", and assume that o is the head of a long list of documents. The program evaluating the query starts from o and follows the links. However, it is possible that while the query program traverses the list, more and more documents are added to its end. If the reading is not faster than the appending, the computation will never terminate and the user will get the impression of dealing with an infinite list. This in spite of the fact that, at each point in time, the set of documents in the Web is finite.

Observe that not all the queries suffer from this non-terminating behavior. Consider the query "list all documents reachable from o by a path of distance less than 2." Suppose it is evaluated by a program that reads o, lists the URL's of all its outgoing links, reads the document for each such URL (if it exists), and repeats the process once for the fetched documents. This program will always terminate, no matter how the Web changes while it runs. Note, however, that the answer may contain mixed data from various states of the Web – some of the documents o was pointing to may have been deleted after being read, some non-existing documents may have been added just before their URL was tested, etc. So the program does not necessarily return an answer that reflects the state of the Web as of some specific point in time.

This section studies the effect of the dynamic environment on queries. In particular, we focus on the effect on the computability and termination of queries. We assume a local-store based architecture where each document is fetched from the Web just once and cached locally if needed. In our Web machines this will be reflected by restricting the oracle to be called at most once on each oid. (So if several accesses are needed, the machine should record the oracle answer on its working tape).

6.1 Databases and queries

We next redefine our data model, and the notions of computable and eventually computable queries. Next we study the properties of queries in this dynamic context.

Definition 6.1 *A* **dynamic Web database schema**, *is a modification of a Web database schema where instead of the single node and link relation schemes, there is an infinite sequence of relation schemes $N_1, L_1, N_2, L_2, \ldots$. An* **instance** *of a dynamic Web database is a mapping associating a finite set of tuples with every relation name in DB, finite sets of node objects with each of the N_i's, and finite sets of link objects with each of the L_i's, so that under the mapping, (DB, N_i, L_i) is a Web database instance, for all $i > 0$.*

We assume that each of the N_i, L_i pairs reflects the state of the Web at some point in time, and that the sequence records the different states of the Web over time. We ignore here the issue of time granularity. Consecutive pairs may be mapped to the same instance to reflect the fact that the Web did not change in that time period, or may differ greatly to reflect the fact that many changes took place.

When a query is computed on the Web, its result may depend on how fast the query is evaluated – the faster it is evaluated, the fewer changes will be reflected in the result. The user often has no control or knowledge of the computation speed, which can depend on many parameters such as network or server load. Thus, at the abstract level, the user can view a query as a nondeterministic mapping. We want our model of computation to capture this nondeterminism and highlight the relationship between computation speed and query result. We incorporate these ideas into our definition of dynamic Web machines as follows.

Definition 6.2 *A* **dynamic Web machine** *is a slightly modified Web machine:*

1. *The oracle input tape is now infinite and contains an encoding of the sequence of relation pairs (N_1, L_1), (N_2, L_2), ...*
2. *The oracle is called at most once on each oid, and*
3. *At the beginning of the computation, the oracle uses (N_1, L_1). On each call it can decide nondeterministically whether to continue answering using the current (N_i, L_i) or switch to (N_{i+1}, L_{i+1}).*

Each computation sequence reflects the relative speeds of computation and Web change. For example, the sequence where all calls are evaluated against (N_1, L_1) corresponds to a very fast computation that manages to complete all access to the Web before any changes take place (or alternatively a computation where concurrency control is employed). A sequence where all the calls are against distinct (N_i, L_i) pairs corresponds to the opposite case, where the rate of Web changes is as fast as the document fetching speed. Sequences where some of the calls are against the same (N_i, L_i), and some against distinct ones, reflect changing relative speeds.

Definition 6.3 *A* **(eventually computable) dynamic Web query**, *is a non-deterministic mapping , i.e. a relation, from dynamic Web database instances to (possibly infinite) sets of tuples over the values in the instances, that is C-generic and (eventually) computable by a dynamic Web machine .*
"Computable by a dynamic Web machine " here means: for each input, all possible computations of the dynamic Web machine terminate, each possible output of the query is computed by some computation, and each computation computes some possible output.
"Eventually computable by a dynamic Web machine " here means: for each input, and each (possibly infinite) output, there is a computation of the machine that eventually computes this output, and every computation eventually computes some output. [5]

[5] *eventually computes* is as in Definition 4.1.

Consider "list all documents reachable from o," and "list all documents reachable from o by a path of length less than 2," discussed at the beginning of this section. Intuitively, the second one can be computed by a dynamic Web machine , where all possible computation sequences terminate. Thus it is a dynamic Web query . On the other hand, any machine for the first one will have some infinite computation sequences, so it is only an EC dynamic Web query . (All this will be made more precise, and formally proved, below).

All the computation sequences of dynamic Web queries always terminate no matter how much the Web changes, or how fast the computation is. This is not the case for EC dynamic Web queries . Nevertheless, some of the EC dynamic Web queries behave more nicely than others. Of particular interest are queries that guarantee computability when computation is relatively fast compared to update rate, as suggested in the following definition.

Definition 6.4 *A* **semi-computable** *dynamic Web query, denoted SC dynamic Web query , is an EC dynamic Web query that can be computed by some dynamic Web machine such that every convergent computation terminates. A convergent computation of a dynamic Web machine is one where there is some k such that the oracle never switches to (N_i, L_i) with $i > k$.*

Intuitively this means the user is guaranteed that, if the query is executed with a sufficiently fast engine, the computation terminates. Note that the engine can start working slowly (so the computation may be affected by changes in the Web), but if at some point it accelerates enough so that all needed data from the Web are retrieved before any more Web changes happen, then the computation is guaranteed to halt. In concurrency control terms this means that such queries can start running without concurrency control – it suffices that concurrency control is employed at *some* point of the computation to guarantee termination.

To clarify things, here are a few examples.

- "List all nodes reachable from o" is an example of a SC dynamic Web query . If at some point the query engine accelerates enough so that all computation is done before any further updates occur, only nodes listed so far plus nodes reachable from them in the current state of the Web are output. Thus the computation terminates.
- On the other hand, "list all the nodes on the Web" is not an SC dynamic Web query , but only an EC dynamic Web query . This is because, even in the static case, when no changes at all are made to the Web (this is the simplest convergent computation), the machine does not halt.
- Finally, as in the static case, a query like "list all nodes with no incoming edges" is neither an SC nor an EC dynamic Web query . Again, this is because even when the Web does not change at all, the query cannot be eventually computed.

6.2 Dynamic vs. Static queries

Query engines for the Web, such as WebSQL[16], W3QL [14], etc., provide query languages to specify user requests. The semantics of such engines is usually defined with respect to a static Web, i.e. what should the query result be if evaluated on some given Web database instance. However, in practice, when the program produced by the query compiler is executed, it runs in a dynamic environment where the Web keeps changing. Speaking in terms of Web machines, this means that the query is compiled into an ordinary (static) Web machine, but, when executed, the oracle in the machine is replaced by a dynamic one, and the dynamic machine is run instead.

What is the relationship between the original (static) query and the actual (dynamic) query being evaluated? This is the question studied below. First we need some auxiliary definitions.

Definition 6.5 *A dynamic Web machine M* **simulates** *a Web machine M' if M is obtained from M' by replacing its oracle by a dynamic one.*

Definition 6.6 *A (EC) dynamic Web query Q* **simulates** *a (EC) Web query Q' if there is a dynamic Web machine M for Q, and a Web machine M' for Q', such that M simulates M'.*

Note that this requires M' to call the oracle on each document only once. This is not a problem since every Web machine can be modified to satisfy this requirement by storing the answers of the oracle on the working tape and checking before each call if data on the given oid is already available.

We are now ready to state the relationship between queries, as defined by the user having a static view of the Web in mind, and the actual queries being executed in the dynamic environment. Not surprisingly, if the Web is not being updated, the actual execution computes exactly what the programmer intended.

Proposition 6.1 *If Q simulates Q', Q returns the same answer as Q' when given the same input and a static Web. That is, on input $(DB, (N_1, L_1), (N_2, L_2), ...)$ where $N_i = N_1, L_i = L_1$ for all i, Q computes a unique result that is the same as that of Q' on (DB, N_1, L_1).*

What happens when the Web is being updated? We focus below of computability issues and explain how they are affected by updates. The first theorem says that computable Web queries are precisely the class of queries whose computation, in a dynamic environment, is guaranteed to terminate provided that the query engine accelerates enough at some arbitrary point in the computation.

Theorem 6.2 *Every SC dynamic Web query simulates some Web query, and every Web query is simulated by some SC dynamic Web query .*

The next theorem deals with eventually computable Web queries. It says that EC Web queries are precisely the queries that can be run in a dynamic environment in a way that guarantees that all tuples in the result will eventually be output.

Theorem 6.3 *Every EC dynamic Web query simulates some EC Web query and every EC Web query is simulated by some EC dynamic Web query .*

We considered above semi-computable and eventually-computable dynamic Web queries . What about the computable ones ?

Theorem 6.4 *Every dynamic Web query simulates some Web query, but not conversely, i.e. there are Web queries that cannot be simulated by any dynamic Web query .*

This means that there are some Web queries such that no program for them is guaranteed to terminate when run in a dynamic environment. Furthermore, in the full paper we show that even if the query programmer is aware of the fact that the query will be run in a dynamic environment, writing an appropriate program that terminates on all inputs may be an impossible task. We do this by establishing the following.

Proposition 6.5 *There is no dynamic Web query Q that, given a static Web as input, computes the reachability set of a node o in the input.*

This shows that some Web queries cannot be simulated by dynamic Web queries , hence they cannot be run in a dynamic environment safely, i.e., with a termination guarantee. Clearly, some queries can (such as "all nodes reachable from o by a path of length smaller then k" for some fixed k). Characterizing the class of Web queries that can run safely is an open problem.

However, in the full paper we present some sufficient conditions. For example, we show that all queries expressed by safe Web calculus formulas where all path expressions are star-free can be simulated by dynamic Web queries .

Thus it is possible to compile any such query into a program that will run safely in a dynamic environment, i.e. with no non-termination concerns. In fact, the programs generated by the implemented WebSQL compiler[16] for these queries are such terminating programs.

7 Forms

So far we ignored for simplicity the issue of forms. We explain below how they fit into the model and extend our results to this more comprehensive model.

7.1 Static Web

As before, we first ignore updates to the Web. We start with the data model for the static case.

Data model

Recall from Section 2 that forms are modeled as documents with parameterized outgoing links. These parameterized links are described in a relation L_p having the same structure as the links relation L, with an additional attribute *param_set*. The exact description of each parameter set is stored in the relation P using tuples of the form $[param\_set: t_1, param\_name: t_2, param\_val: t_3]$. We assume that the domain of set id's and of possible parameter names and values is infinite. So a Web database has the form $W = (DB, N, L, L_p, P)$.

An **instance** of a Web database is W a mapping associating a finite set of tuples with each of the relations, such that the following holds: (1) *id* is a key in relation N, (2) there is a referential integrity constraint between the *source* attribute in L and L_P, and the *id* attribute in N, and (3) there is a referential integrity constraint between the *params_set* attribute in P to the one in L_p.

Queries and eventually computable queries

Queries and eventually computable queries were were defined above using Web machines. We slightly modify the oracle in the machine to support parameterized links:
(1) The oracle input tape now also contains an encoding of the relations L_p and P.
(2) When called, the oracle expects to find on its working tape a sequence of the form *oid#plist*, where *plist* is either empty or is a sequence of $pnam_1\#pval_1\#...\#pnam_n\#pval_n$.
(3) If *plist* is empty, the oracle works as before and returns the (encoded) node tuple from N, and the tuples in L describing its outgoing links. Otherwise, the oracle returns the (encoded) node tuple from N, and the L_p tuples of its outgoing links such that the associated parameter set (according to P) is exactly the one described by *plist*.

(EC) Web queries are now defined as before, except that the Web instances are now over the extended schema, and the Web machine being used is the extended one described above. Also, note that the values of the attribute *params_set* are used only to encode sets of parameters, and are not really part of the data. We thus consider as (EC) Web queries only the mappings from Web instances to sets of tuples over the values in the instance, not including this attribute.

Before we reconsider the properties of (EC) Web queries in this extended framework, we need to refine the definitions of the equivalence ($=_S$) and the order ($\leq_S$) relations among Web instances, which were used in Sections 3 and 4 resp. to characterize the queries. The refined definition is almost the same, except that now node reachability also has to take into consideration the parameterized links, and is thus omitted.

Once this is done we can restate the result for Web queries.

Theorem 7.1 *Theorem 3.2 (using the refined definitions) still hold in the presence of forms.*

Thus, even in the presence of forms, Web queries are precisely the queries that are only interested in the sub-Web reachable using data in the base relations.

For the case of EC Web queries we further need to assume that for each form, the set of possible parameter names and values potentially leading to actual documents is computable, given the form. This seems a reasonable assumption since in practice they can often be determined from the form structure and the size of attributes in it.

Theorem 7.2 *Under the above assumption, Theorem 4.1 (using the refined definitions) still hold in the presence of forms.*

The second characterization we had for (EC) Web queries was through (monotone) Web relational machines. This characterization also carries over to the case of forms, with some slight extension to the definition of the machines. The assumption of computability of form attributes, for the case of EC Web queries, is needed here as well.

7.2 Dynamic Web

In the presence of forms, a dynamic Web database contains a sequence of relations $N_1, L_1, L_{p_1}, P_1, N_2, L_2, L_{p_2}, P_2, ...$, and the oracle in the dynamic Web machine switches between them. Dynamic Web queries, EC dynamic Web queries and SC dynamic Web queries are defined as before using this machine. The relationship between queries in the static and dynamic environment is stated in terms of simulation. All the results presented above still hold (sometimes under specific restrictions):

Theorem 7.3 *Theorems 6.2 and 6.4 still hold in the presence of forms.*

Theorem 7.4 *Under the assumption of computability for form attributes, Theorem 6.3 still holds in the presence of forms.*

8 Conclusion and further work

We have given a new model of query and computation on the Web, closely related to, but significantly different from, the model of [2]. We have separated the two reasons for non-terminating behavior of Web queries: navigational access to data and lack of concurrency control, and analyzed their impact.

Cost is an important aspect of query evaluation. One of the contributions of our computation model is in providing a good starting point for a theory of query complexity in the Web context. The conventional approach in database theory is to estimate query evaluation time as a function of the size of the database. Viewing the Web as infinite, as done in [2], makes such an estimation problematic. Even if the Web is viewed as a finite database, simply adapting the standard complexity measures does not seem adequate – in the Web context, it is not realistic to try to evaluate queries whose complexity would be considered feasible in the usual theory, such as polynomial or even linear time.

For a query to be practical, it should not attempt to access too much of the network. Query analysis thus involves, in this context, two tasks: (1) Estimation of what part of the network may be accessed by the query. In terms of our computation model, this means estimating how many oracle calls are needed to evaluate a query, as a function of the size of the base relations, the size of the Web, and perhaps some bounds on Web parameters such as maximal size of documents, maximum outdegree, etc. (2) The cost of the query can then be analyzed in traditional ways as a function of the size of the base relations plus the size of the sub-network being accessed.

Note that this separation lets us to take into consideration not only how much of the Web is accessed, but also the cost of accessing it. With the current Web architecture, the cost of document access is affected by document properties (e.g. size) and by the cost of communication between the site where the query is being evaluated and the site where the document is stored. This can be modeled by pricing oracle calls according to these parameters.

Preliminary work on complexity of Web-queries in the context of a static Web was presented in [16], where the notion of *query locality* was introduced to characterize the size of the network being accessed. Syntactic restrictions on WebSQL queries were presented to capture various locality classes. The case of EC queries and dynamic Web still needs to be studied.

Acknowledgements

We thank David Toman for his comments on an earlier version.

References

[1] S. Abiteboul, C.H. Papadimitriou, and V.Vianu. The power of the reflective relational machine. In *Proc. IEEE Symp. on Logic in Computer Science*, 1994.

[2] S. Abiteboul and V. Vianu. Queries and computation on the Web. In *Proc. ICDT '97*, 1997.

[3] S. Abiteboul and V.Vianu. Generic computation and its complexity. In *Proc. ACM STOC*, pages 209–219, 1991.

[4] Serge Abiteboul, Sophie Cluet, and Tova Milo. Querying and updating the file. In *Proceedings of the 19th VLDB Conference*, 1993.

[5] P. Atzeni, G. Mecca, P. Merialdo, and E. Tabet. Structures in the Web. Technical Report RT-INF-19-1997, Dip. di Informatica e Automazione, Università di Roma Tre, 1997.

[6] C. Beeri and Y. Kornatzky. A logical query language for hypertext systems. In *Proc. of the European Conference on Hypertext*, pages 67–80. Cambridge University Press, 1990.

[7] P. Brown and M. Stonebraker. BigSur: a system for the management of earth science data. In *Proceedings of VLDB '95*, pages 720–28, 1995.

[8] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of SIGMOD '96*, pages 505–516, 1996.

[9] Ashok K. Chandra and David Harel. Computable queries for relational data bases. *JCSS*, 21(2):156–178, 1980.

[10] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proc. ACM SIGMOD'94*, pages 313–324, 1994.

[11] M. P. Consens and A. O. Mendelzon. Expressing structural hypertext queries in Graphlog. In *Hypertext'89*, pages 269–292, 1989.

[12] Digital Equipment Corporation. Alta Vista: Main page. 1996. `http://altavista.digital.com`.

[13] Ralf Hartmut Güting, Roberto Zicari, and David M. Choy. An algebra for structured office documents. *ACM TOIS*, 7(2):123–157, 1989.

[14] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proc. of VLDB'95*, pages 54–65, 1995.

[15] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A declarative language for querying and restructuring the Web. In *Proc. of 6th. International Workshop on Research Issues in Data Engineering, RIDE '96*, New Orleans, February 1996.

[16] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proc. PDIS '96*, pages 80–91, December 1996.

[17] T. Minohara and R. Watanabe. Queries on structure in hypertext. In *Foundations of Data Organization and Algorithms, FODO '93*, pages 394–411. Springer, 1993.

[18] Jan Paredaens, Jan Van den Bussche, Marc Andries, Marc Gemis, Marc Gyssens, Inge Thyssens, Dirk Van Gucht, Vijay Sarathy, and Lawrence V. Saxton. An overview of GOOD. *SIGMOD Record*, 21(1):25–31, 1992.